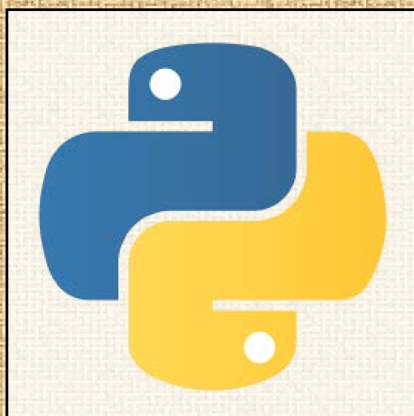
Escuela Superior Politécnica del Litoral

Python
Programación



Libro digital

Versión 2.0 – 2015

Luis Rodríguez Ojeda

Python Programación

Prefacio

Este documento es una contribución bibliográfica para los estudiantes que toman un primer curso de Programación de Computadoras a nivel universitario. El estudio del material incluido en los primeros doce capítulos no tiene pre-requistos, solamente el interés en conocer un lenguaje actual que posteriormente pueda ser usado como el soporte para resolver computacionalmente problemas de diferente nivel de complejidad en ingeniería, matemáticas y otras áreas. Sin embargo, es deseable que los interesados tengan algun conocimiento básico de la lógica matemática.

El enfoque didáctico utilizado en este documento es el aprendizaje mediante ejemplos y desarrollo de ejercicios propuestos. El material incluye muchos ejemplos para describir los conceptos algorítmicos en forma práctica y su traducción al lenguaje computacional Python.

Python es un interpretador de instrucciones muy eficiente y de acceso libre y público disponible para su instalación desde la red internet. El lenguaje Python es fácil de aprender y aplicar, versátil y muy conveniente para iniciar el aprendizaje de lenguajes de programación de manera progresiva y creativa usando diferentes metodologías de programación.

El soporte de este documento es la experiencia desarrollada por el autor impartiendo cursos de enseñanza de lenguajes de programación para estudiantes de ingeniería y el haber desarrollado otros documentos digitales de apoyo bibliográfico.

Este documento es de uso público y distribución libre y se adhiere a la corriente de desarrollar textos digitales que puedan ser actualizados y mejorados continuamente y disponibles para su uso en línea, reduciendo el consumo de papel y tinta, contribuyendo así con el cuidado del medio ambiente.

El documento ha sido compilado en un formato que facilita el uso de la información. Se puede controlar el tamaño del texto en pantalla, agregar un índice electrónico para facilitar búsqueda de temas, resaltar digitalmente texto, insertar comentarios, notas, enlaces, revisiones, etc. y que no sería posible en un texto impreso.

Escuela Superior Politécnica del Litoral
Luis Rodríguez Ojeda, M.Sc.
Profesor
2014
lrodrig@espol.edu.ec

Organización del material

El capítulo **1** establece un modelo general para la resolución de problemas con el computador. Los usuarios pudieran darle unos pocos minutos a su lectura.

Los capítulos **2** y **3** son opcionales. Pueden ser de interés para los usuarios que quieran entender los conceptos abstractos de algoritmos y la construcción de algoritmos computacionales independientemente de un lenguaje de programación específico.

El capítulo **4** tiene información general acerca de lenguajes de programación y metodologías de programación. Su lectura tomará pocos minutos

Los capítulos **5**, **6**, **7**, **8**, y **9** contienen el material para conocer y practicar el lenguaje de programación Python. Su estudio cubriría el tiempo de un semestre académico para estudiantes de carreras de ingeniería.

Los capítulos **10**, **11** y **12** son una introducción a temas que normalmente son de interés para estudiantes que siguen una carrera orientada a áreas computacionales.

Los capítulos **13** y **14** pueden ser de interés para usuarios que tienen un nivel de conocimientos matemáticos más avanzados y requieren resolver problemas matemáticos de este tipo con el soporte de librerías especializadas de Python.

Contenido

Python Programación

1 Introducción

1.1 Objetivo y requisitos

Esta obra es una contribución para el desarrollo de una metodología computacional para resolver problemas basada en los principios de la construcción de algoritmos.

El soporte computacional es el lenguaje Python con el que se explora y se adquiere la práctica y el conocimiento suficiente para la programación de computadoras aplicada a la resolución de problemas matemáticos, de ingeniería y otras ciencias. Es deseable que los interesados tengan algun conocimiento básico de la lógica matemática.

1.2 Metodología

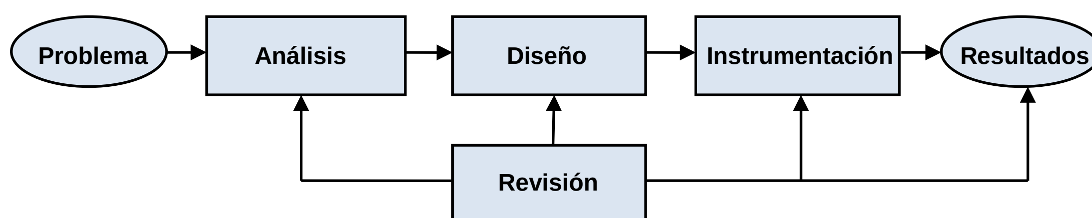
Mediante explicaciones basadas en ejemplos el usuario puede adquirir en forma progresiva los conocimientos necesarios para resolver problemas y su programación en Python. El desarrollo de variados ejercicios proporcionará la base para extenderlos y aplicarlos a la resolución de problemas más complejos. Al mismo tiempo, el usuario podrá desarrollar su propio estilo de programación.

1.3 Un modelo para resolver problemas con el computador

El análisis y diseño de soluciones computacionales es una ciencia que facilita el uso eficiente del poder de las computadoras para resolver problemas.

Para facilitar el desarrollo de estas soluciones, es adecuado usar un lenguaje computacional simple, general y eficiente como el que ofrece Python.

El siguiente gráfico describe los pasos en la solución computacional de un problema



Primero, es necesario asegurarnos que el problema que intentamos resolver está en nuestro ámbito de conocimiento. No es recomendable intentar resolver problemas si no tenemos el conocimiento y la práctica para enfrentar su solución.

En la etapa de **Análisis** se estudia el problema en forma detallada: sus características, las variables y los procesos que intervienen. Asimismo, se deben definir los datos que se requieren y cual es el objetivo esperado. El resultado de esta etapa son las **especificaciones** detalladas de los requerimientos que en algunos casos se pueden expresar en forma matemática.

En la etapa de **Diseño** se procede a elaborar los procedimientos necesarios para cumplir con los requerimientos especificados en el análisis, incluyendo fórmulas, tablas, etc. El objeto resultante se denomina **algoritmo**.

En la etapa de **Instrumentación,** si el problema es simple, se puede obtener la solución interactuando directamente mediante instrumentos disponibles en el entorno computacional. Si el problema es más complejo, deben construirse **programas** y definir el ingreso y la organización de los datos necesarios.

Al concluir la etapa de la instrumentación, se usan datos para realizar **pruebas** de los programas. Es necesario que se realice una revisión en cada etapa de este proceso y que se validen los resultados obtenidos antes de aceptarlos y continuar.

Posteriormente se efectúa la instalación y operación. Debe preverse la necesidad de mantenimiento y cambios en los programas para ajustarlos al entorno en el que se usarán.

Este proceso necesita ser planificado y sistematizado para que su desarrollo sea eficiente siendo imprescindible seguir normas, utilizar metodologías de programación y mantener una documentación adecuada.

Los instrumentos computacionales modernos tales como Python disponen de facilidades para probar interactivamente instrucciones y programas a medida que son desarrollados, mejorando así la productividad. También ofrece librerías que facilitan el desarrollo de los proyectos de programación.

2 Algoritmos

Un algoritmo es una descripción ordenada de las instrucciones que deben realizarse para resolver un problema en un tiempo finito.

Para crear un algoritmo es necesario conocer en forma detallada el problema, las variables, los datos que se necesitan, los procesos involucrados, las restricciones, y los resultados esperados o por lo menos los criterios para considerarlos correctos.

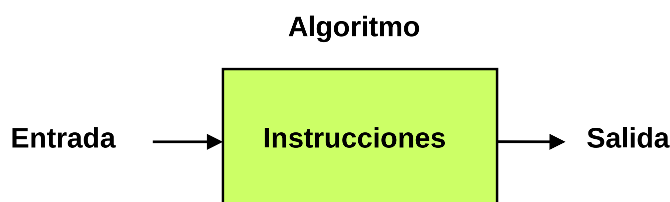
La descripción del algoritmo debe orientarse a la instrumentación computacional, pero cuando los problemas son simples, puede omitirse la elaboración detallada del algoritmo e ir directamente a la codificación en el lenguaje computacional siempre que este ofrezca una sintaxis fácil de usar y entender.

Es muy importante desarrollar el pensamiento algorítmico progresivamente con la ayuda de ejemplos y con la práctica. El objetivo final es estructurar una metodología para facilitar la resolución de problemas, investigando sus componentes y cumpliendo los requerimientos.

2.1 Estructura de un algoritmo

Un algoritmo es un objeto que debe comunicarse con el entorno. Por lo tanto debe incluir facilidades para el ingreso de datos y la salida de resultados.

Dentro de un algoritmo se describe el procedimiento, mediante instrucciones, que realizará la transformación de los datos y producirá los resultados esperados.



2.2 Lenguajes para escribir algoritmos

Para escribir algoritmos se pueden usar diferentes notaciones: lenguaje natural, lenguajes gráficos, lenguajes simbólicos, etc.

Para que una notación sea útil debe poseer algunas características que permitan producir algoritmos fáciles de construir, entender y aplicar:

1) Las instrucciones deben ser simples para facilitar su uso.
2) Las instrucciones deben ser claras y precisas para evitar ambigüedades.
3) Debe incluir suficientes instrucciones para describir la solución de problemas.
4) Preferentemente, las instrucciones deben tener orientación computacional.

Los algoritmos deben ser reproducibles, es decir que al ejecutarse deben entregar los mismos resultados si se utilizan los mismos datos.

2.3 Definiciones

a) Proceso

Conjunto de acciones realizadas al ejecutar las instrucciones descritas en un algoritmo.

b) Estado

Situación de un proceso en cada etapa de su realización, desde su inicio hasta su finalización. En cada etapa, las variables pueden modificarse.

c) Variables

Símbolos con los que se representan los valores que se producen en el proceso.

Componentes de una variable:

Nombre:	Identificación de cada variable
Tipo:	Conjunto de valores o dominio asociado a la variable
Contenido:	Valor asignado a una variable
Celda:	Dispositivo que almacena el valor asignado a una variable

2.4 Introducción a la construcción de algoritmos

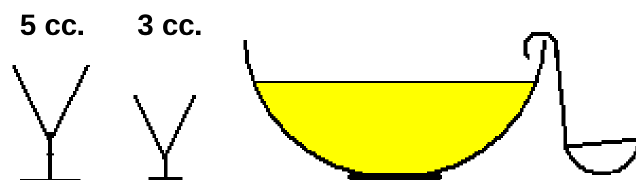
El desarrollo de algoritmos constituye una metodología para resolver problemas en forma organizada.

Primero debe definirse el objetivo al que se desea llegar, luego deben identificarse los componentes o variables y escribir las instrucciones necesarias para poder realizar los cambios que permitirán llegar al objetivo propuesto. Para validar el algoritmo debe realizarse alguna prueba con la cual se puede verificar que el resultado es correcto.

A continuación se propone un problema y se describe un procedimiento algorítmico para llegar a la solución.

Ejemplo. En el gráfico siguiente se muestra un recipiente grande con algún refresco y se propone el problema de obtener exactamente **4 cc.** usando solamente los instrumentos mostrados los cuales tienen indicada su capacidad. Se supondrá que el recipiente grande contiene suficiente cantidad de refresco. Es posible trasladar el contenido entre recipientes pero no se dispone de ningún dispositivo adicional para medición.

Describa un algoritmo para llegar a la solución.



Objetivo propuesto: Que el recipiente de **5 cc.** contenga **4 cc.** del refresco



Variables

Sean **A**: Representación del recipiente cuya capacidad es **5 cc.**
B: Representación del recipiente cuya capacidad es **3 cc.**
C: Representación del recipiente grande con cantidad suficiente de refresco.

Algoritmo

1. Llene **A** con el refresco de **C**
2. Vierta **A** en **B** hasta llenarlo
3. Vierta todo el contenido de **B** en **C**
4. Vierta el resto del contenido de **A** en **B**
5. Llene **A** con el refresco de **C**
6. Vierta el contenido de **A** en **B** hasta llenarlo
7. El recipiente **A** contendrá **4 cc**.

Prueba

Recorrer el algoritmo anotando los valores que toman las variables **A** y **B**

Instrucción	Contenido de A	Contenido de B
Inicio	0	0
1	5	0
2	2	3
3	2	0
4	0	2
5	5	2
6	4	3

Resultado

Se verifica que en el recipiente **A** quedarán **4 cc.**

Observe los componentes que intervienen en la construcción del algoritmo:

a) Propuesta del objetivo
b) Definición de variables
c) Lista de instrucciones
d) Prueba del algoritmo
e) Verificación del resultado obtenido

Ejemplo. Describa un algoritmo para revisar un vehículo antes de un viaje.

Algoritmo

1 **Si el nivel de agua del radiador está bajo**
    Complete el nivel de agua del radiador
2 **Si el nivel de gasolina es bajo**
    Acuda a la estación de gasolina y llene el tanque
3 **Si el nivel de aceite del motor es bajo**
    Acuda a la estación de servicio para chequear el vehículo
4 **Para cada llanta repita la siguiente instrucción**
    Compruebe la presión del aire
5 **Si alguna llanta registró presión baja**
    Acuda a la estación de servicio para revisión de llantas

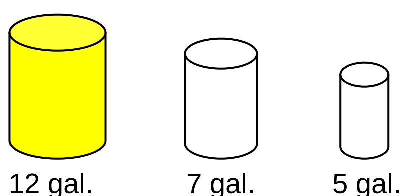
Esta descripción pretende ser un algoritmo para la revisión del vehículo. Contiene **acciones condicionadas** y también una instrucción para **repetir una acción** varias veces. Sin embargo, el uso del lenguaje común no permite que la descripción sea suficientemente precisa para facilitar el seguimiento de las instrucciones. Tampoco se puede constatar que se cumple el objetivo propuesto como en el ejemplo anterior. Por lo tanto, se lo puede considerar simplemente como un instructivo de ayuda.

Los lenguajes algorítmicos deben ser permitir crear descripciones claras, de tal manera que no haya posibilidad de interpretar las instrucciones de más de una manera.

2.5 Ejercicios de creación de algoritmos

Para cada ejercicio proponga un algoritmo para obtener la solución.

1. Se tienen 3 recipientes cilíndricos, opacos y sin marcas, de 12, 7, y 5 galones de capacidad. El recipiente de 12 galones está lleno de combustible. El objetivo es repartir el combustible en dos partes iguales usando únicamente los tres recipientes. Considere que puede trasladar el combustible entre recipientes pero no se dispone de algún instrumento de medición.



a) Describa gráficamente el resultado esperado
b) Asigne símbolos a las variables (Representan la cantidad de combustible)
c) Construya un algoritmo para obtener la solución. Numere las instrucciones
d) Ejecute las instrucciones y registre los cambios del contenido de las variables
e) Verifique que el algoritmo produce la solución esperada.

Para probar su algoritmo puede completar una tabla como la siguiente. Suponga que A, B, C representan a los recipientes con la capacidad y en el orden dados en el gráfico anterior.

Instrucción	A	B	C
Inicio	12	0	0
1			
2			
. . .			

Nota: Existe una solución en 12 pasos (en cada paso se traslada de un recipiente a otro).

2. Describa un algoritmo para resolver el siguiente conocido problema. Defina las variables, escriba y numere las instrucciones y luego efectúe una prueba para verificar que funciona**:**

Tres misioneros y tres caníbales deben atravesar un río en un bote en el que sólo caben dos personas. Pueden hacer los viajes que quieran, pero en en las orillas y en el bote el número de caníbales no debe ser mayor al de los misioneros porque ya podemos suponer lo que ocurriría. El bote no puede cruzar el río si no hay al menos una persona dentro para que lo dirija.

Sugerencia: Defina los misioneros como **M1, M2, M3** y los caníbales como **C1, C2, C3.** Las variables **R1, R2** son las orillas del río y **B** el bote. El contenido de estas variables cambiará mediante las instrucciones del algoritmo. Después de construir el algoritmo puede completar una tabla como la siguiente para verificar el resultado:

Instrucción	R1	B	R2
Inicio	**M1,M2,M3,C1,C2,C3**		
1			
2			
. . .			
Final			**M1,M2,M3,C1,C2,C3**

3. Describa un algoritmo para resolver el siguiente problema, también muy conocido. Defina las variables, escriba y numere las instrucciones y luego efectúe una prueba para verificar que funciona**:**

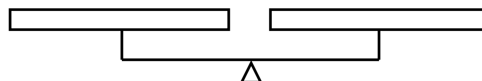
Había un pastor que cuidaba a un lobo, una oveja y una canasta de lechugas. El pastor tenía que cruzar un río, para lo cual disponía de un pequeño bote en el que solamente cabían él y un animal, o él y la canasta de lechugas. El problema es conseguir que pasen todos al otro lado del río sanos y salvos, sin que nadie se coma a nadie. Al lobo no le gustan las lechugas, pero como se puede suponer, el lobo no puede quedarse a solas con la oveja y tampoco la oveja puede quedarse sola con las lechugas. El pastor debe guiar al bote en cada viaje.

Sugerencia: Defina símbolos para los datos **P**: pastor, **L**: lobo, **O**: oveja, **C**: canasta**.** Las variables **R1, R2** son las orillas del río y **B** el bote. El contenido de estas variables cambiará mediante las instrucciones del algoritmo. Después de construir el algoritmo puede completar una tabla como la siguiente para verificar el resultado:

Instrucción	R1	B	R2
Inicio	**P, L, O, C**		
1			
2			
. . .			
Final			**P, L, O, C**

4. Describa un algoritmo para resolver el siguiente problema. Defina las variables, escriba y numere las instrucciones y luego efectúe una prueba para verificar que funciona**:**

Se tiene una caja con nueve bolas, semejantes en apariencia, entre las cuales hay una más pesada que las otras ocho. No se sabe cuál es y se trata de hallarla efectuando solamente dos pesadas en una balanza de dos platillos en equilibrio.



Después de construir el algoritmo puede completar una tabla como la siguiente para verificar el resultado, en donde a, b, c, d, e, f, g, h, i representan a las nueve bolas.

Instrucción	Caja	Platillo izquierdo	Platillo derecho
Inicio	**a, b, c, d, e, f, g, h, i**		
1			
2			
. . .			
Final			

5. Describa en forma precisa las instrucciones necesarias para preparar una fiesta sorpresa para su amiga o su amigo. En las instrucciones debe incluir los días y horas en los que serán desarrolladas las actividades. Haga referencia a la fecha y hora cero en la que ocurrirá el evento. Verifique su algoritmo mediante un cuadro con fechas y horas. En este cuadro anote el desarrollo de las actividades siguiendo las instrucciones de su algoritmo. Note que este tipo de algoritmos no se puede verificar que cumplen el objetivo propuesto como en los ejercicios anteriores. Pueden considerarse únicamente como instructivos para organizar el desarrollo de actividades.

3 Construcción de algoritmos computacionales

En esta sección se describirá una notación para construir algoritmos computacionales. La notación es suficientemente simple y clara para ser usada en problemas básicos facilitando la construcción de su solución, contribuyendo además al desarrollo del pensamiento algorítmico y la lógica de programación de computadoras. Esta notación es útil especialmente en la etapa inicial de aprendizaje de la programación. Posteriormente se puede prescindir de ella.

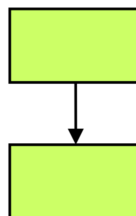
Un algoritmo es la descripción ordenada de la idea que se propone para resolver un problema. Esta idea debe desarrollarse identificando los componentes que permitirán llegar a solución. Cada componente puede incluir una instrucción simple o un conjunto de instrucciones.

Para desarrollar una metodología representaremos cada componente gráficamente mediante un bloque:



Bloque para representar componentes del algoritmo

Un algoritmo puede contener varios componentes que son ejecutados en forma secuencial. Este orden se lo puede indicar explícitamente mediante líneas de flujo que unen los bloques**:**

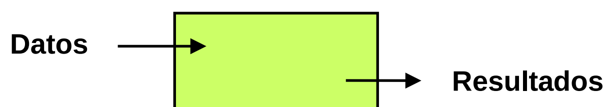


Bloque con el primer componente

Bloque con el segundo componente

3.1 Instrucciones u operaciones elementales

Los componentes de un algoritmo contienen instrucciones u operaciones con las que se especifican cálculos y otros procesos. Si el algoritmo debe comunicarse con el entorno entonces debe incluir instrucciones para la entrada de datos y la salida de los resultados.

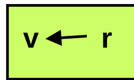


Las flechas que entran o salen del bloque representan la interacción del algoritmo con el exterior.

Simbología

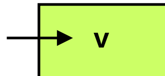
Sean **v** una variable y **r** algún valor que se desea usar en el algoritmo.

a) Instrucción de asignación



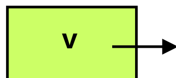
Asigna dentro del bloque el valor **r** a la variable **v**

b) Instrucción de entrada



Ingresa un valor desde afuera del bloque para la variable **v**

c) Instrucción de salida



Muestra fuera del bloque el valor que contiene la variable **v**

Es una buena práctica documentar la construcción del algoritmo. El algoritmo y cada variable utilizada deberían tener alguna descripción.

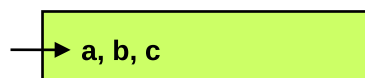
Es preferible que los datos para las variables sean ingresados al algoritmo desde fuera del bloque. De esta manera el algoritmo se independiza de los datos y no hay que cambiar las instrucciones dentro del bloque para realizar pruebas con nuevos datos.

Ejemplo. Describa un algoritmo para calcular el área de un triángulo conocidos sus tres lados.

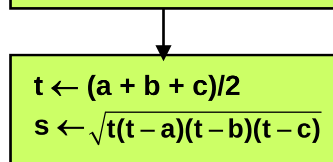
Algoritmo: Área de un triángulo

Variables

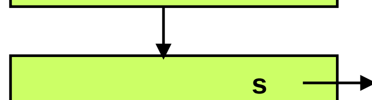
a, b, c: Lados del triángulo (Datos desconocidos)
s: Área del triángulo (Es el resultado esperado)
t: semiperímetro (Valor usado para la fórmula del área)
$s = \sqrt{t(t-a)(t-b)(t-c)}$, (Fórmula del área del triángulo)
siendo **t = (a + b + c)/2**



Bloque de entrada de datos



Bloque de cálculos



Bloque de salida de resultados

Realice algunas pruebas del algoritmo anterior. En cada prueba ingrese los datos necesarios desde fuera del algoritmo:

Prueba	a	b	c	t	s
1	5	6	8	9.5	14.9812
2	4	7	6	8.5	11.9765
3	8	6	9	11.5	23.5252

El algoritmo produce los resultados esperados para los datos dados en cada prueba.

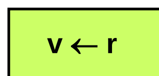
3.2 Diagramas de flujo

Si se usan símbolos especiales con orientación computacional, el resultado es un objeto muy conocido, un diagrama de flujo para describir algoritmos. Esta notación será usada en varios ejemplos de este documento.

Símbolos de dagramas de flujo

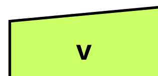
Sean **v** una variable y **r** algún valor que se desea usar en el algoritmo.

a) Instrucción de asignación



Asigna el valor **r** a la variable **v**

b) Instrucción de entrada



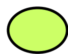
Entrada manual de datos (por teclado) para la variable **v**

c) Instrucción de salida



Muestra en pantalla el valor que contiene la variable **v**

d) Inicio y final del diagrama de flujo



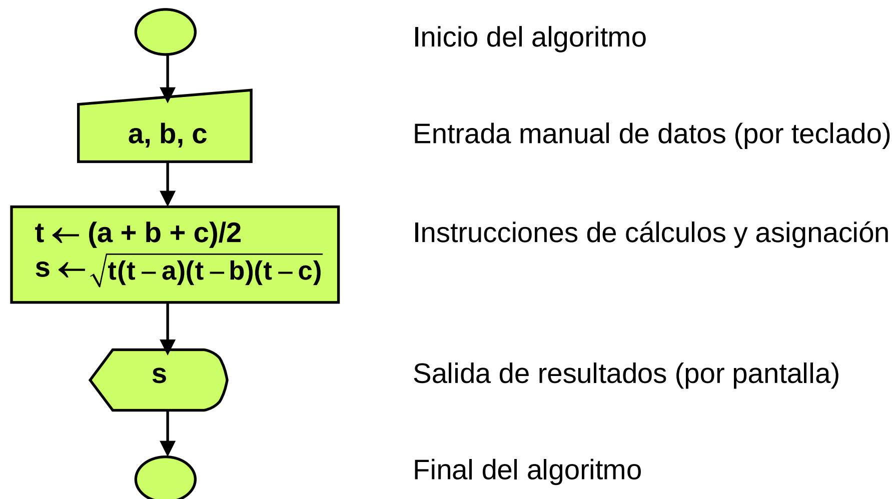
e) Líneas de flujo



Existen símbolos adicionales para describir otras operaciones computacionales

Ejemplo. Describa mediante un diagrama de flujo la solución para el ejemplo del triángulo

Diagrama de flujo



Realice algunas pruebas del algoritmo anterior. En cada prueba ingrese los datos necesarios desde fuera del algoritmo:

Prueba	a	b	c	t	s
1	5	6	8	9.5	14.9812
2	4	7	6	8.5	11.9765
3	8	6	9	11.5	23.5252

El algoritmo produce los resultados esperados para los datos dados en cada prueba.

Los bloques y las líneas de flujo de los diagramas ayudan a la comprensión de la lógica de los algoritmos, especialmente cuando se deben incluir instrucciones cuya ejecución está condicionada o cuando se necesita describir la ejecución repetida de bloques de instrucciones. Esta notación gráfica se usará para describir las instrucciones básicas del lenguaje computacional y en algunos ejemplos iniciales.

Con la práctica y cuando se familiariza con la lógica algorítmica, se podrá prescindir del dibujo de bloques y líneas de flujo y escribir el algoritmo mediante algún **seudo lenguaje** y posteriormente directamente con un **lenguaje de programación.**

3.3 Seudo lenguaje

Los seudo lenguajes no tienen reglas fijas para escribir instrucciones pero deberían tener la claridad suficiente para expresar el algoritmo en forma precisa y estructurada usualmente orientada a su futura instrumentación computacional.

Simbología

Sean **v** una variable y **r** algún valor que se desea usar en el algoritmo.

a) Instrucción de asignación

v ← r Asigna el valor **r** a la variable **v**

b) Instrucción de entrada

Entrar v Ingresa un valor para la variable **v** desde afuera del algoritmo

c) Instrucción de salida

Mostrar v Muestra fuera del algoritmo el valor que contiene la variable **v**

El color es solamente para resaltar la acción que se desea realizar con el algoritmo

Ejemplo. Describa en seudo lenguaje la solución para el ejemplo del triángulo

Entrar a, b, c
$t \leftarrow (a + b + c)/2$
$s \leftarrow \sqrt{t(t-a)(t-b)(t-c)}$
Mostrar s

Realice algunas pruebas del algoritmo anterior. En cada prueba ingrese los datos necesarios desde fuera del algoritmo:

Prueba	a	b	c	t	s
1	5	6	8	9.5	14.9812
2	4	7	6	8.5	11.9765
3	8	6	9	11.5	23.5252

El algoritmo produce los resultados esperados para los datos dados en cada prueba.

En el diseño y construcción de un algoritmo, lo más importante es el conocimiento del problema y la concepción de la idea para resolverlo.

La notación algorímica es solamente un instrumento para expresar de manera estructurada la idea propuesta con la que se espera llegar a la solución del problema.

3.3.1 Algunas instrucciones típicas de asignación en notación algorítmica

Los siguientes ejemplos se proponen para explicar la notación y algunos aspectos del uso de las instrucciones de asignación en los algoritmos.

a) Asigne a la variable **n** la raiz cuadrada de **5.**

$\mathbf{n} \leftarrow \sqrt{5}$

b) Asigne a la variable **s** el valor **0**.

s ← 0

Normalmente se inician variables con cero para luego agregar valores. Es costumbre denominar a estas variables con el nombre de **"acumulador"** o **"sumador".**

Algunas variables también se inician con cero para luego incrementarlas con un valor unitario. Estas variables se usan para conteos y se las distingue con el nombre de **"contador".**

c) Modifique el valor actual de la variable **s** incrementándolo con el valor de **u**.

s ← s + u

Si las variables **u** o **s** no tuviesen asignadas algún valor previo, será un error.
En esta instrucción la misma variable **s** aparece a la izquierda y a la derecha.
La asignación modifica el valor de esta variable.
Es importante distinguir la **asignación algorítmica** de la **igualdad que se usa en el lenguaje matemático** en el cual sería incorrecto escribir: **s = s + u**

d) Modifique el valor actual de la variable **k** incrementándolo en **1**.

k ← k + 1

Si **k** no ha sido asignada previamente, será un error.

e) Modifique el valor de la variable **r** reduciendo su valor actual en **2**

r ← r – 2

f) Modifique el valor de la variable **n** duplicando su valor actual

n ← 2n

g) Modifique el valor de la variable **x** incrementando su valor actual en **20%**

x ← 1.2 x

h) Modifique el valor de la variable **t** reduciendo su valor actual en **5%**

t ← 0.95 t

j) Intercambie el contenido de las variables **a** y **b**

v ← a
a ← b
b ← v

Se requiere usar una variable adicional para no perder uno de los valores.

Es un error realizar la asignación de la siguiente manera**:**

a ← b
b ← a

Se perdería el valor que contenía la variable **a**

Cada variable puede contener un solo valor en cualquier momento de la ejecución del algoritmo. Este valor es el que ha sido asignado más recientemente.

Algunos ejemplos contienen la misma variable a la izquierda y a la derecha. La variable a la derecha contiene el valor actual. Con este valor se realiza alguna operación y el resultado es asignado a la variable que también aparece a la izquierda. Este último valor es el que conserva la variable al continuar el algoritmo.

Este tipo de asignación es usado frecuentemente en los algoritmos pues permite cambiar el contenido de las variables durante la ejecución.

3.3.2 Ejercicios con la notación algorítmica: Algoritmos secuenciales

Para cada ejercicio escriba una solución en notación algorítmica (diagrama de flujo o seudo lenguaje) y realice una prueba

1. Dados el radio y altura de un cilindro calcule el área total y el volumen

2. Se tiene un recipiente cilíndrico con capacidad en litros. Su altura es un dato en metros. Determine el diámetro de la base

3. Dadas las tres dimensiones de un bloque rectangular calcule y muestre su área total y su volumen

4. La siguiente fórmula proporciona el enésimo término **u** de una progresión aritmética:

u = a + (n − 1) r

en donde **a** es el primer término, **n** es el la cantidad de términos y **r** es la razón entre dos términos consecutivos. Calcular el valor de **r** dados **u, a, n**

5. El examen de una materia es el 70% de la nota total. Las lecciones constituyen el 20% y las tareas el 10% de la nota total. Ingrese como datos la nota del examen calificado sobre 100 puntos, la nota de una lección calificada sobre 10 puntos, y las notas de tres tareas calificadas cada una sobre 10 puntos. Calcule la calificación total sobre 100 puntos.

3.4 Estructuras de control de flujo de un algoritmo

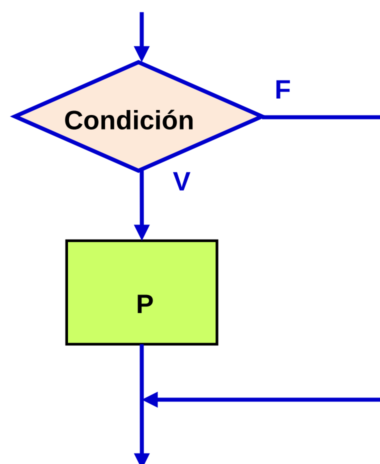
La ejecución de los bloques de un algoritmo es secuencial de arriba hacia abajo, pero este orden puede alterarse mediante estructuras de control de flujo que permiten establecer un orden especial en la ejecución. Para describirlas se usará una representación gráfica.

3.4.1 Decisones

Describen la ejecución selectiva de bloques usando como criterio el resultado de una condición.

a) Ejecución condicionada de un bloque

Representación gráfica



Al entrar a esta estructura, se evalúa la condición. Si el resultado es verdadero (**V**) se ejecutarán las instrucciones en el bloque **P** caso contrario, si el resultado es falso (**F**) el bloque no será ejecutarlo. En ambos casos el algoritmo continua debajo del bloque.

La condición es cualquier expresión cuyo resultado puede ser únicamente verdadero (**V**) o falso (**F**). Puede incluir operadores para comparar el contenido de variables y también se pueden usar los conectores de la lógica matemática.

Seudo lenguaje

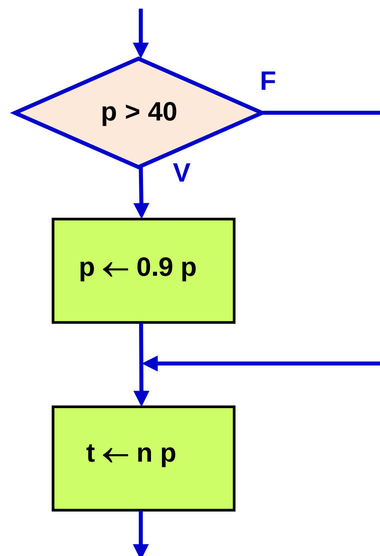
Si **condición**
  P
Fin

Ejemplo. Expresiones que pudieran ser usadas como una condición

$n > 0$
$a \leq 5$
$x \neq 4$
$a < 3 \vee x > 1$

Para que una expresión pueda evaluarse y ser usada como una condición, las variables incluidas en la expresión deben tener asignado algún valor, caso contrario será un error pues la condición no podría evaluarse.

Ejemplo. Describa en notación algorítmica la acción de reducir en **10%** el valor que contiene la variable **p**, en caso de que su valor actual sea mayor a **40**. Despues obtenga el resultado de la multiplicación de **n** por el valor de **p** (con su valor inicial o con su valor corregido).



Antes del bloque, la variable **p** debe haber sido asignada con algún valor, caso contrario sería un error.

Ejemplo. Describa en notación algorítmica una solución al siguiente problema.

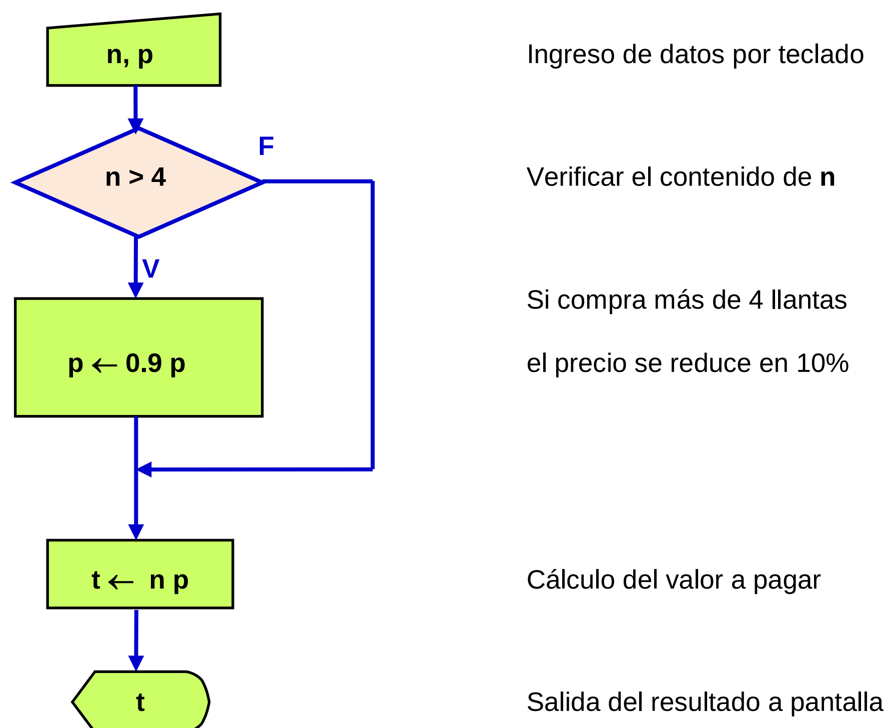
Calcular el valor total que una persona debe pagar por la compra de llantas en un almacén que tiene la siguiente promoción: Si la cantidad de llantas comprada es mayor a **4**, el precio unitario tiene un descuento de **10%.** El algoritmo debe ingresar como datos la catidad de llantas y el precio inicial de cada llanta. Mediante una comparación el algoritmo deberá aplicar el descuento.

Algoritmo: Compra de llantas con descuento

Variables

- **n**: Cantidad de llantas
- **p**: Precio inicial de cada llanta
- **t**: Valor a pagar

Diagrama de flijo



Prueba. Realice algunas pruebas del algoritmo anterior. En cada prueba ingrese los datos necesarios desde fuera del bloque:

Pruebas	n	p	t	Salida
1	2	70	140	140
2	6	80	432	432
3	5	120	540	540

El algoritmo produce los resultados esperados con los datos de cada prueba

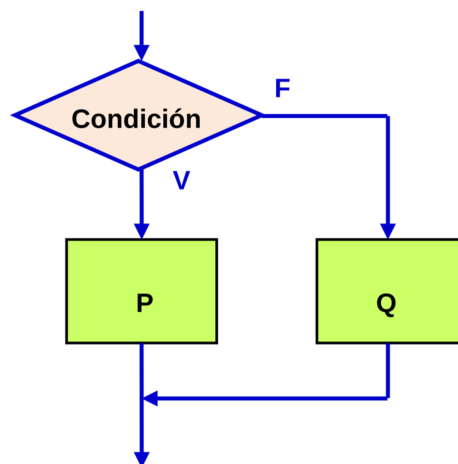
Describa el algoritmo del ejemplo anterior en seudo lenguaje

```
Entrar n,p
Si n>4
   p ← 0.9 p
Fin
t ← np
Mostrar t
```

El color es solamente para resaltar la acción que se desea realizar en el algoritmo

b) Ejecución selectiva entre dos bloques

Representación gráfica

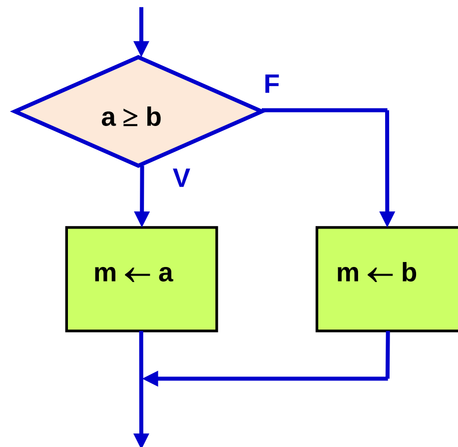


Al entrar a esta estructura, se evalúa la condición. Si el resultado es verdadero (**V**) se ejecutará el bloque **P** asociado al valor verdadero, caso contrario, si el resultado es falso (**F**) se ejecutará el bloque **Q**. El algoritmo continua abajo, despues de ejecutar alguno de los dos bloques.

Seudo lenguaje

```
Si  condición
       P
Sinó
       Q
Fin
```

Ejemplo. Describa en notación algorítmica como asignar a la variable **m** el mayor entre dos valores almacenados respectivamente en las variables **a** y **b**



Antes del bloque, las variables **a** y **b** deben haber sido asignadas algún valor, caso contrario sería un error.

Ejemplo. Describa en notación algorítmica una solución al siguiente problema.

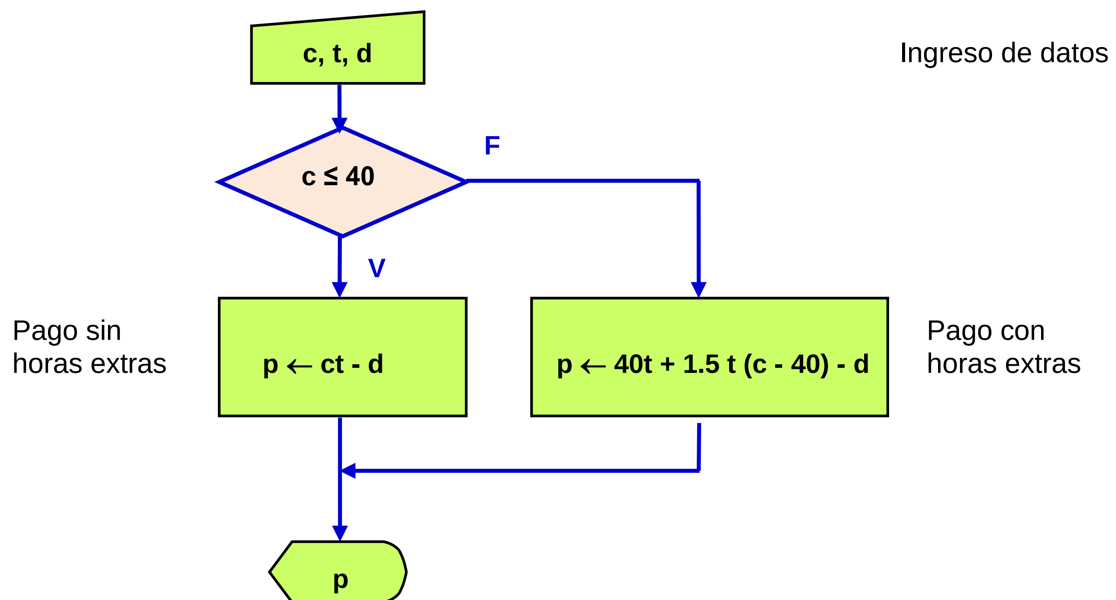
Para el pago semanal a un obrero se consideran los siguientes datos: horas trabajadas, tarifa por hora y descuentos. Si la cantidad de horas trabajadas en la semana es mayor a 40, se le debe pagar las horas en exceso con una bonificación de 50% adicional a la tarifa normal.

Algoritmo: Pago semanal a un obrero

Variables

c: Cantidad de horas trabajadas en la semana
t: Tarifa por hora
d: Descuentos que se aplican al pago semanal
p: Pago que recibe el obrero

Diagrama de flujo



Prueba. Realice algunas pruebas del algoritmo anterior. En cada una ingrese los datos necesarios desde fuera del bloque.

Pruebas	c	t	d	p	Salida
1	40	5	20	180	180
2	35	4	25	115	115
3	42	5	30	185	185

El algoritmo produce los resultados esperados con los datos de cada prueba

Describa el algoritmo del ejemplo anterior en seudo lenguaje

```
Entrar c,t,d
Si c ≤ 40
   p ← ct - d
Sinó
   p ← 40t + 1.5 t (c - 40) - d
Fin
Mostrar p
```

3.4.2 Ejercicios con la notación algorítmica: Algoritmos con decisiones

Para cada ejercicio desarrolle una solución en notación algorítmica y realice una prueba

1. Dados el radio y altura de un cilindro, si la altura es mayor al radio calcule y muestre el valor del volumen del cilindro, caso contrario muestre el valor del área del cilindro.

2. Lea la cantidad de Kw que ha consumido una familia y el precio por Kw. Si la cantidad es mayor a 700, incremente el precio en 5% para el exceso de Kw sobre 700. Muestre el valor total a pagar.

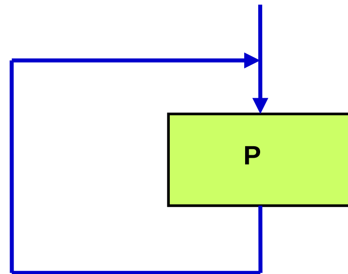
3. Lea un valor de temperatura **t** y un código **p** que puede ser **1** o **2**. Si el código es **1** convierta la temperatura **t** de grados **f** a grados **c** con la fórmula **c=5/9(t-32).** Si el código es **2** convierta la temperatura **t** de grados **c** a **f** con la fórmula: **f=32+9t/5.** Muestre el resultado.

4. Dadas las dimensiones de un bloque rectangular, calcule las diagonales de las tres caras diferentes. Muestre el valor de la mayor diagonal.

5. Dadas las tres calificaciones de un estudiante, encuentre y muestre la calificación mas alta.

3.4.3 Ciclos

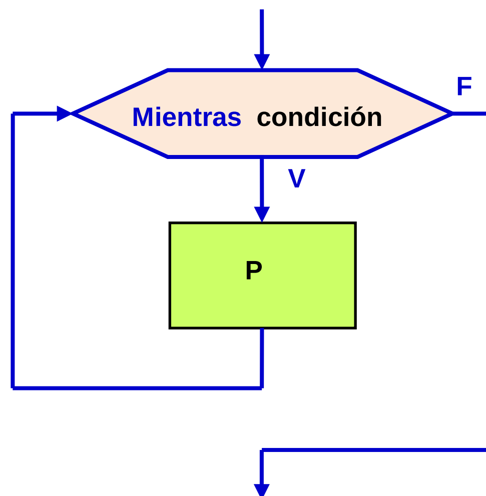
Estas estructuras de control se usan para describir la ejecución repetida de un bloque de instrucciones. El objetivo es colocar el bloque de instruciones dentro de un ciclo como se muestra en el gráfico. Sin embargo, es necesario agregar un dispositivo que permita salir del ciclo para que el algoritmo pueda continuar:



Hay tres formas comunes que se usan para salir de una estructura de repetición. Dos de ellas usan una condición para salir del ciclo. Esta condición puede estar al inicio o al final. La otra forma, utiliza los valores de un conteo o una secuencia de valores para controlar la repetición.

a) Ejecución repetida de un bloque mediante una condición al inicio

Representación gráfica



Al entrar a esta estructura se evalúa la condición. Si el resultado es verdadero (**V**) se ejecutarán las instrucciones dentro del bloque y regresará nuevamente a evaluar la condición.

Mientras la condición mantenga el valor verdadero (**V**), el bloque de instrucciones se ejecutará nuevamente. Esto significa que en algún ciclo al evaluar la condición deberá obtenerse el resultado falso (**F**) para salir de la estructura y continuar la ejecución después del bloque. Al diseñar el algoritmo deberán escribirse las instrucciones necesarias.

La condición es cualquier expresión cuyo resultado puede ser únicamente verdadero (**V**) o falso (**F**). Puede incluir operadores para comparar el contenido de variables y también se pueden usar los conectores de la lógica matemática.

Ejemplo de expresiones que pudieran ser usadas como una condición. El resultado de cada una dependerá del contenido de las variables:

$n > 0$
$a \leq 5$
$x \neq 4$
$a < 3 \vee x > 1$

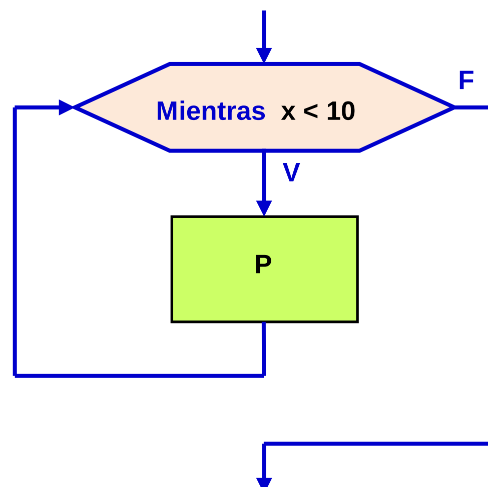
Seudo lenguaje

Mientras condición

P

Fin

Ejemplo. Describa algorítmicamente la repetición de un bloque de instrucciones mientras la variable **x** tenga un valor menor a **10**



Antes del bloque, la variable **x** debe haber sido asignada con algún valor, caso contrario sería un error.

Es necesario que la variable **x** cambie su contenido dentro del bloque de instrucciones que se repiten para que en algún ciclo la repetición pueda terminar y la ejecución continue después del bloque. Caso contrario sería un error lógico pues el algoritmo permanecería en el ciclo.

NOTA: Los símbolos gráficos usados en este documentos para representar ciclos no son símbolos estandarizados, pero son suficientemente descriptivos. Una notación más conocida usa rombos para describir las repeticiones condicionadas, sin embargo esto puede agregar alguna ambigüedad al graficar el algoritmo

Ejemplo. Describa en notación algorítmica una solución para el siguiente problema.

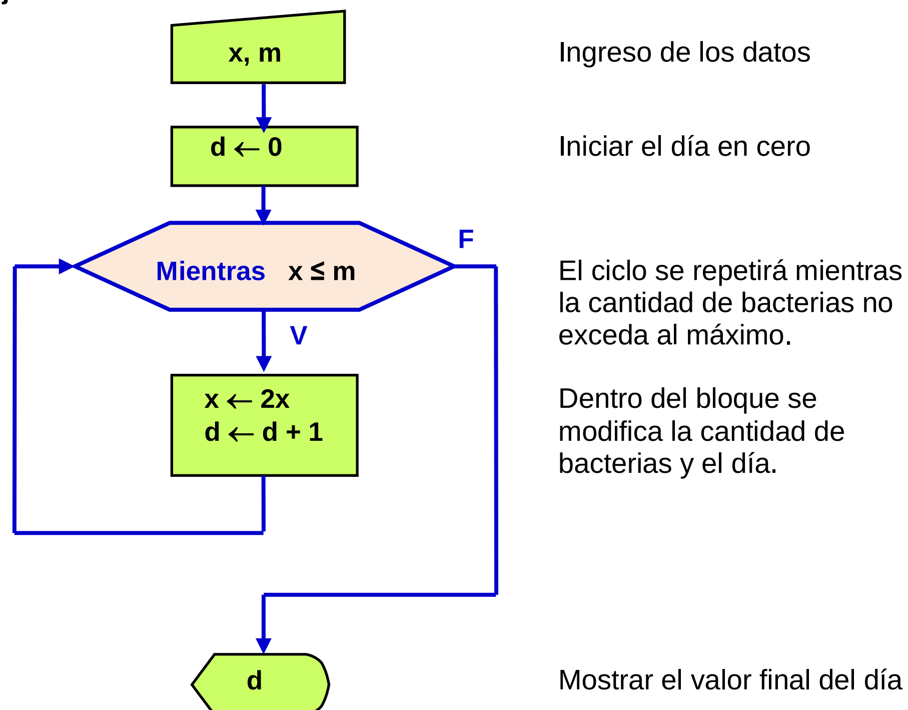
En un cultivo se tiene una cantidad inicial de bacterias. Cada día esta cantidad se duplica. Determine en que día la cantidad excede a un valor máximo.

Algoritmo: Crecimiento de la cantidad de bacterias

Variables

x: Cantidad inicial de bacterias
m: Cantidad máxima de bacterias
d: Día

Diagrama de flujo



Note que se debe usar el ciclo condicionado pues no se puede anticipar la cantidad de repeticiones necesarias para que la cantidad de bacterias exceda al valor máximo.

Prueba. Realice una prueba del algoritmo anterior. Ingrese los datos desde fuera del bloque y registre los cambios en el contenido de las variables.

Prueba	x	m	d	Salida
	200	5000	0	
	400		1	
	800		2	
	1600		3	
	3200		4	
	6400		5	5

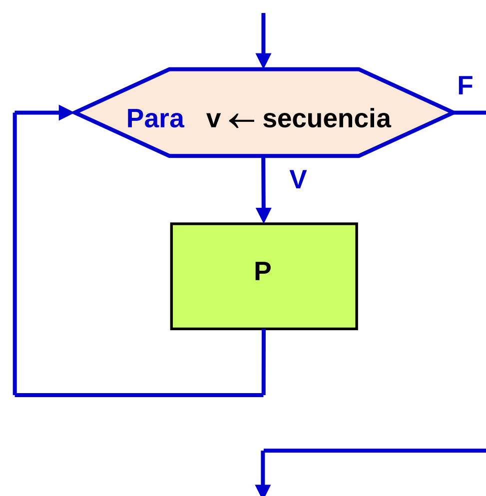
Se puede verificar que es el resultado esperado y que coincide con el valor que se puede calcular matemáticamente con la expresión $2^n(x) > m$

Describa el algoritmo del ejemplo anterior en seudo lenguaje

```
Entrar x,m
d ← 0
Mientras  x ≤ m
   x ← 2x
   d ← d + 1
Fin
Mostrar d
```

b) Ejecución repetida de un bloque mediante una secuencia de valores

Representación gráfica



Para usar esta estructura de control es necesario especificar una variable para el conteo de repeticiones y una lista de valores o secuencia que puede tomar. El ciclo se repetirá con cada valor especificado para la variable. Al ejecutarse cada ciclo el valor de la variable cambiará siguiendo la especificacion.

Al entrar a esta estructura, se inicia la variable de control. Si esta variable no excede al valor final, se ejecuta el bloque y regresa nuevamente al inicio del ciclo y la variable toma el siguiente valor de la secuencia. Cuando el valor de la variable llegue al valor final, el ciclo finalizará y la ejecución continuará después del bloque.

Seudo lenguaje

```
Para  v ← secuencia

      P

Fin
```

La variable de control del ciclo puede especificarse con alguna notación que exprese cuales son los valores que puede tomar.

Definición de la secuencia de valores

Mediante una lista de valores:

Ejemplo. **n ← [2, 5, 4, 7, 6]**

Indicando el valor inicial, el valor final de la secuencia y el incremento:

Ejemplo. **k ← 1, 10, 1**

Genera la secuencia: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

Ejemplo. **i ← 1, 15, 2**

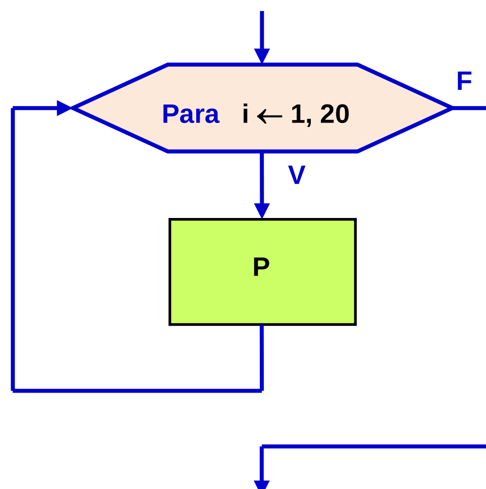
Genera la secuencia: **1, 3, 5, 7, 9, 11, 13, 15**

Si no se escribe el incremento, se supondrá que es la unidad.

Ejemplo. **k ← 1, 10**

Genera la secuencia: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

Ejemplo. Especificar una variable de control para que el bloque de instrucciones **P** se repita **20** veces



Ejemplo. Describa en notación algorítmica una solución al siguiente problema.

Dado un entero positivo **n**, se desea verificar que la suma de los primeros **n** números impares es igual a n^2

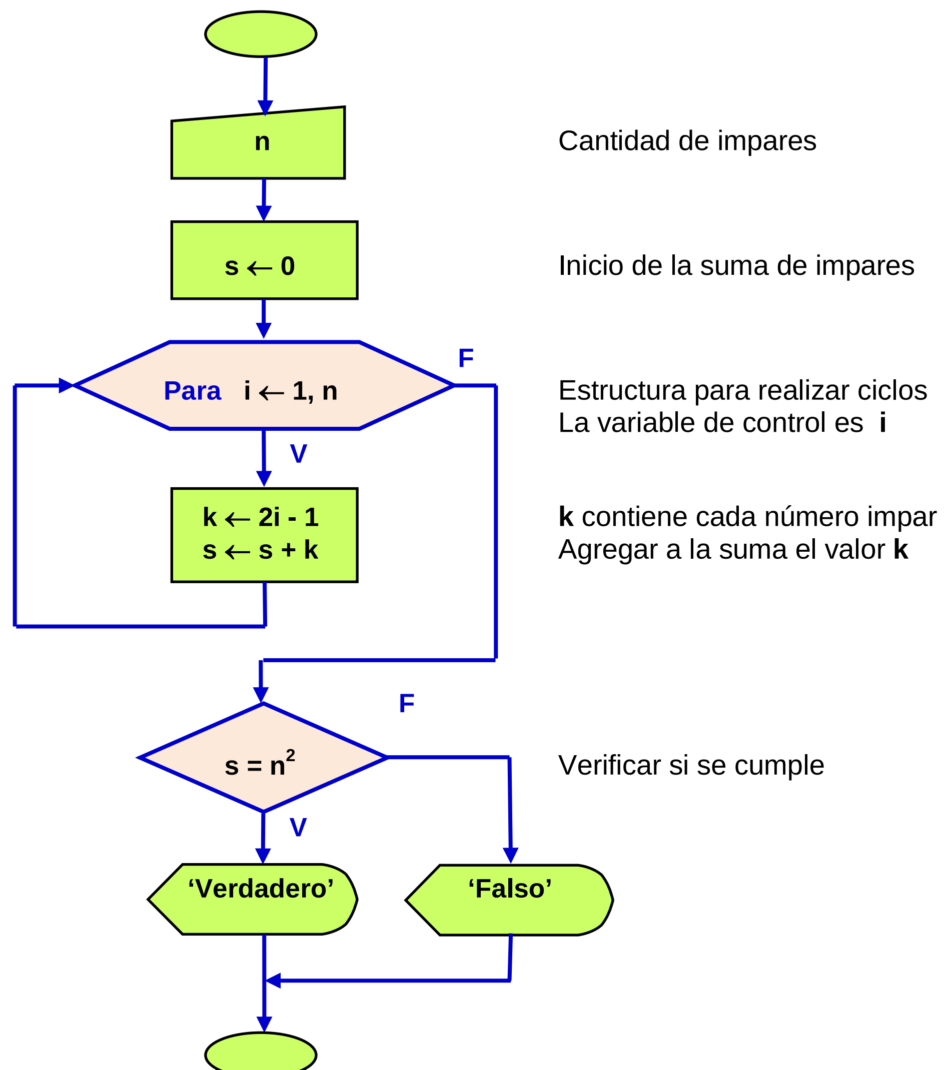
Ej. n = 5: 1 + 3 + 5 + 7 + 9 = 5^2

Algoritmo: Suma de impares

Variables

n: Cantidad de números impares
k: Cada número impar
s: Suma de impares
i: Conteo de ciclos

Diagrama de flujo



Prueba. Realice una prueba del algoritmo anterior. Ingrese un dato desde fuera del bloque y registre los cambios en el contenido de las variables.

Prueba	**n**	**Ciclo i**	**Impar k**	**s**	**Salida**
	5			0	
		1	1	1	
		2	3	4	
		3	5	9	
		4	7	16	
		5	9	25	**'Verdadero'**

Se verifica que el resultado es **'Verdadero'.**

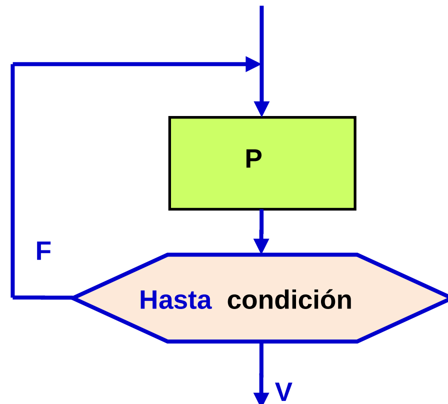
Note que este algoritmo no constituye una demostración matemática. Solo verfica que la propiedad se cumple para algunos valores específicos.

Describa el algoritmo del ejemplo anterior en seudo lenguaje

```
Entrar n
s ← 0
Para  i ← 1, 2, 3, ..., n
   k ← 2i - 1
   s ← s + k
Fin
Si  s = n²
   Mostrar 'Verdadero'
Sinó
   Mostrar 'Falso'
Fin
```

c) Ejecución repetida de un bloque mediante una condición al final

Algunos programadores prefieren definir ciclos con la condición al final. Esta estructura se usa para **repetir** un bloque **hasta** que se cumpla alguna condición. Se la puede representar con el siguiente gráfico:

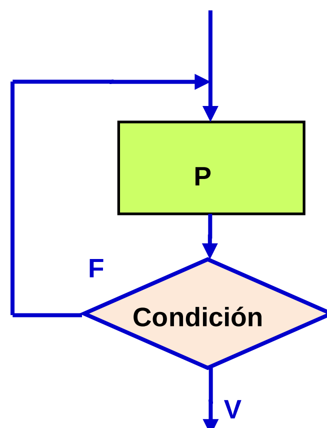


Al entrar a esta estructura, se ejecutan las instrucciones en el bloque y después se chequea el valor de la condición. Si la condición tiene el valor verdadero, termina el ciclo y la ejecución continua abajo, caso contrario, nuevamente se repite el bloque. En esta estructura algorítmica, el bloque de instrucciónes **se repite al menos una vez**.

Seudo lenguaje

```
Repita
    P
Hasta  condición
```

Nota: El símbolo gráfico usado para describir las estructuras de repetición en este documento no son estandarizadas pero hemos elegido usar estos símbolos distintivos. Algunos programadores prefieren usar el rombo de las decisiones para describir repeticiones condicionadas pero esto introduce alguna ambigüedad en el gráfico del algoritmo. Se muestra esta representación gráfica tipicamente usada:



Las estructuras de repetición son necesarias cuando un bloque del algoritmo debe ejecutarse más de una vez para construír la solución.

Si se puede anticipar la cantidad de ciclos que se deben realizar, entonces conviene usar la repetición controlada con un conteo de ciclos definido con una secuencia.

Si el algoritmo que se propone para resolver un problema requiere repetir un bloque pero no se puede anticipar la cantidad de ciclos que deben realizarse, entonces deberia usarse la repetición controlada con una condición.

3.4.4 Ejercicios con la notación algorítmica: Algoritmos con ciclos

Para cada ejercicio desarrolle una solución en notación algorítmica y realice una prueba

1. Calcule el mayor valor de los pesos de **n** paquetes en una bodega. Estos datos ingresan uno a la vez dentro de un ciclo. Al inicio ingrese el valor de **n** para especificar la cantidad de ciclos que se realizarán

2. Lea los votos de **n** personas en una consulta. Cada voto es un número **0,** o **1** correspondiente a la opción a favor (**1**) o en contra (**0**). Al inicio lea el valor de **n** para especificar la cantidad de ciclos que se realizarán. Muestre el resultado de la consulta.

3. Determine la suma de los **n** primeros números de la serie: **1, 1, 2, 3, 5, 8, 13, 21, ...** en la cual cada término, a partir del tercero, se obtiene sumando los dos términos anteriores

4. Calcule un valor aproximado para la constante **π** usando la siguiente expresión:

$$\pi/4 = 1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + 1/13 \ldots$$

La cantidad de términos es un dato que debe ser ingresado al inicio del algoritmo.

5. Determine la cantidad de términos que deben sumarse de la serie $1^1 + 2^2 + 3^3 + 4^4 + \ldots$ para que el valor de la suma sea mayor a un número **x** ingresado al inicio.

6. El inventor del juego del ajedréz pidió a su rey que como recompensa le diera por la primera casilla 2 granos de trigo, por la segunda, 4 granos, por la tercera 8, por la cuarta 16, y así sucesivamente hasta llegar a la casilla 64. El rey aceptó. Suponga que cada Kg. de trigo consta de 20000 granos de trigo. Si cada tonelada tiene 1000 Kg. describa un algoritmo para calcular la cantidad de toneladas de trigo que se hubiesen necesitado.

En el ciclo describa la suma $2^1 + 2^2 + 2^3 + 2^4 + \ldots + 2^{64}$

4 Lenguajes de Programación de Computadoras

Un lenguaje de programación es un lenguaje diseñado para describir acciones que puedan ser realizadas por una computadora. Para que esto sea posible es necesario que se cumplan ciertos requerimietos básicos:

a) Debe haberse construido o elaborado el algoritmo con el procedimiento detallado para resolver el problema de interés. Este algoritmo debe tener orientación computacional.

b) Se necesita conocer la sintaxis y significado de las instrucciones del lenguaje de programación que se va a utilizar

c) Es necesario tener acceso a un equipo computacional, el cual debe tener instalado el traductor del lenguaje computacional que será utilizado.

Los lenguajes de programación están formados por un conjunto de símbolos y reglas para su uso. La instrumentación del algoritmo mediante estas reglas se denomina programar y el producto obtenido es el programa. El programa escrito se denomina programa fuente, mientras que el programa traducido al lenguaje que entiende el computador se denomina programa objeto o ejecutable.

La actividad de programación incluye varias etapas: escritura del programa, pruebas, depuración, validación de los resultados y documentación del desarrollo.

La programación requiere precisión y cuidado en los detalles de uso del lenguaje. Los errores de sintaxis normalmente los detecta el traductor del lenguaje, pero los errores de contenido requieren mayor esfuerzo y la realización de pruebas. Esta etapa se denomina depuración.

En laa aplicaciones reales los programas pueden necesitar cambios y actualizaciones, por lo que el uso de normas y una buena documentación son escenciales.

4.1 Metodologías de programación

Las siguientes son algunas de las metodologías de programación más comunes utilizadas en programación de computadoras: Programación Estructurada, Programación Modular, Programación Orientada a Objetos

La Programación Estructurada usa instrucciones de control básicas con el objetivo de mantener la claridad en la codificación de un programa o módulo. Esta metodología enfatiza el desarrollo de la programación al nivel de detalle.

La Programación Modular divide el desarrollo de programas, usualmente complejos o grandes, en subprogramas o módulos que facilitan la codificación y la validación. Esta metodología orienta la organización de la programación mediante la construcción de bloques o módulos y su integración.

La Programación Orientada a Objetos permite a los programadores organizar el diseño de un proyecto de programación definiendo objetos relacionados con el problema que se intenta modelizar. En esta metodología las estructuras de los datos son el centro alrededor del cual se desarrolla la programación.

En este documento se desarrollará el aprendizaje de la programación con el lenguaje Python aplicando estas metodologías sucesivamente en el orden indicado. Al final, cada usuario podrá reconocerlas y agregará su propio estilo de programación.

4.2 Factores para elegir un lenguaje de programación

Algunos aspectos que deben considerarse al elegir un lenguaje de programación:

- **a)** Propósito o aplicación: general o dedicado
- **b)** Características operativas: interacción, tipos de datos, control de excepciones
- **c)** Soporte técnico: disponibilidad, sistemas operativos, documentación, comunidad
- **d)** Metodologías de programación aplicables
- **e)** Facilidad y tiempo para el aprendizaje
- **f)** Legibilidad de la codificación
- **g)** Eficiencia de los programas resultantes
- **h)** Productividad: velocidad de desarrollo de los programas
- **i)** Perspectivas a futuro para desarrollo y crecimiento del lenguaje

Se dice que un programador debe conocer al menos dos lenguajes de programación. El primero puede ser Python y el siguiente C++, Java, etc. Sin embargo Python puede ser el primero y único lenguaje computacional que necesitan la mayoría de usuarios.

4.3 Lenguajes compilados y lenguajes interpretados

Los compiladores como C++ tienen una ventana para el desarrollo de la programación en la cual se escribe el programa. El programa se compila y si no hay errores se genera un programa ejecutable autónomo. No se requiere que el traductor esté presente. El programa ejecutable resultante es muy eficiente. En los lenguajes compilados solamente pueden realizarse pruebas cuando el programa está completo.

Los interpretadores de lenguaje como Python tienen una ventana interactiva para pruebas y una ventana de edición para crear los programas. Este entorno permite realizar pruebas durante el desarrollo, con lo cual aumenta la productividad.

Los resultados se muestran en la ventana interactiva por lo que se requiere que el interpretador de instrucciones esté presente. Los programas resultantes no son muy eficientes, pero el desarrollo y pruebas es muy rápido. Si se desea más eficiencia se los puede trasladar a un lenguaje compilado.

Actualmente existen compiladores para traducir directamente programa construidos con lenguajes interpretados. Igualmente, se pueden incorporar componentes compilados a un lenguaje interpretado.

5 El lenguaje Python

5.1 Origen del lenguaje Python

El lenguaje Python fue creado por Guido van Rossum a principos de los años 90 en Holanda. El nombre del lenguaje proviene de la afición de su creador por el grupo de humoristas británicos los Monty Python.



Guido van Rossum nació en 1956 en Holanda. Allí recibió un título de maestría en matemática y ciencias de la computación de parte de la Universidad de Amsterdam en 1982. Esto le abrió las puertas a varios puestos de trabajo en los siguientes años en el Centrum Wiskunde & Informatica (CWI). En esa misma ciudad creó Python. Actualmente colabora en Google.

Se lo conoce actualmente por el título BDFL ("Benevolent Dictator for Life'"), teniendo asignada la tarea de fijar las directrices sobre la evolución de Python, así como la de tomar decisiones finales sobre el lenguaje que todos los desarrolladores acatan. Van Rossum tiene fama de ser bastante conservador, realizando pocos cambios al lenguaje entre versiones sucesivas, intentando mantener siempre la compatibilidad con versiones anteriores. (*)

Python es un interpretador de instrucciones que permite usar el lenguaje en forma interactiva. Los lenguajes interpretados, a diferencia de los lenguajes compilados, permiten experimentar interactivamente en una ventana y también mediante programas que pueden desarrollarse y probarse a medida que son construidos. Esta interacción facilita el aprendizaje del lenguaje y mejora la productividad. Los programas compilados en cambio, deben estar completos para que sean probados y no admiten experimentar separadamente con las instrucciones. La ventaja de los programas compilados es que el tiempo de ejecución es menor.

Python es un lenguaje de propósito general. Su diseño no obliga a los usuarios a adoptar un estilo particular. Esta característica del lenguaje motiva la creatividad y permite la elección entre varios paradigmas o metodologías de programación.

Con todos los recursos del lenguaje y el soporte de las librerías disponibles, el programador puede usar libremente su imaginación para crear nuevas soluciones. Partiendo de un conocimiento inicial básico, puede avanzar en el aprendizaje del lenguaje a su propio ritmo.

Python es un producto público y de distribución libre que puede descargarse de internet.

(*) Los datos biográficos se tomaron de la dirección de internet:
http://www.ecured.cu/index.php/Guido_van_Rossum

En la siguiente dirección de YouTube hay un video en el cual Guido van Rossum expone algunos aspectos del lenguaje Python:
http://www.youtube.com/watch?v=EBRMq2Ioxsc

5.2 Caracteristicas del lenguaje computacional Python

a) Python es un lenguaje interpretado. Se considera sucesor del lenguaje ABC y usa conceptos de otros lenguajes como Modula-3, Lisp, entre otros.

b) Python no obliga a los programadores a adoptar un estilo particular de programación.

c) Se puede instalar en varias plataformas: Windows, Linux, etc. Con menores cambios puede trasladarse entre ellas.

d) Es software libre y de código abierto con licencia GPL (General Public License). Se puede instalar, modificar y distribuir proporcionando el código fuente. Una licencia GPL no ofrece garantía, pero la gran comunidad de usuarios que disponen del código abierto, rápidamente detectan errores.

e) El código escrito en Python es legible, sin marcas para definir bloques como en otros lenguajes. No requiere símbolos de fin de línea. Escribir en este lenguaje es casi como escribir en seudo código en inglés

Ejemplo.

```
Si e no está en [12,34,25,17,24]                    Seudolenguaje
      imprimir "No está en la lista"

if e not in  [12,34,25,17,24]:                      Lenguaje Python
      print("No está en la lista")
```

f) Usa el encolumnamiento de instrucciones para definir bloques y establecer el alcance de las estructuras de control. El código resultante es compacto pero legible.

g) Python es un lenguaje de tipado dinámico: conecta un método y un nombre de variable durante la ejecución del programa.

h) Es un lenguaje seguro. Realiza chequeo dinámico de tipos de datos y de índices

i) Es un lenguaje extendible. Continuamente la gran comunidad de usuarios está creando nuevas librerías en diferentes áreas de aplicación.

j) Python es un lenguaje de propósito general. Algunas aplicaciones son:

a) Aprendizaje de la programación.
b) Desarrollo de prototipos.
c) Computación científica y matemática.
d) Estadística y optimización.
e) Desarrollo de interfaz visual.
f) Desarrollo de aplicaciones web.
g) Interfaz para manejo de bases de datos.
h) Educación y juegos.
i) Desarrollo de software.

k) Algunos usuarios actuales de Python: YouTube, Google, NASA, universidades, etc.

l) Aprendizaje fácil para diferentes niveles de usuarios. Recomendable como primer lenguaje de programación.

m) Python puede tener abiertas varias ventanas interactivas y varias ventanas de edición trabajando en pruebas o proyectos diferentes

n) En Python la ejecución se inicia desde la ventana de edición o cargando el programa en la ventana interactiva. Siendo un interpretador de lenguaje, el traductor debe estar presente. Sin embargo ya existen utilitarios para compilar programas Python. Ejemplo. PyPy.

o) En Python, la mayoría de las funciones están en librerías que deben ser cargadas antes de acceder a las funciones. Por este motivo, el tiempo de carga del traductor Python es menor.

p) Python es un lenguaje de propósito general y tiene estructuras de datos muy flexibles. Se pueden crear listas con componentes de diferentes tipos.

La filosofía de Python es la legibilidad y la transparencia. Estos principios están codificados y se los puede revisar escribiendo la instrucción **import this** en la ventana principal de Python.

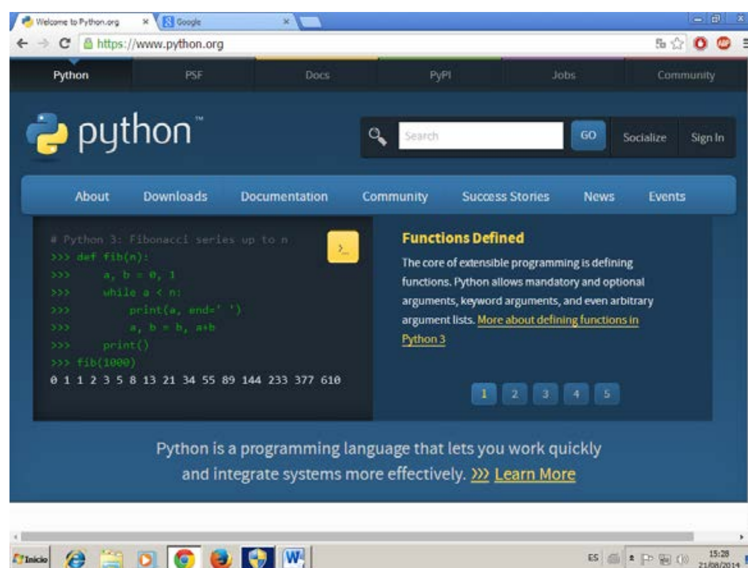
Alguna información para esta sección fue tomada del curso en la dirección de internet:

http://codigofacilito.com/cursos/Python

5.3 Carga e instalación

El traductor del lenguaje de programación Python es público y de acceso libre. Se lo puede descargar e instalar para WINDOWS y para otros sistemas operativos desde la página oficial de Python en la dirección de la red internet:

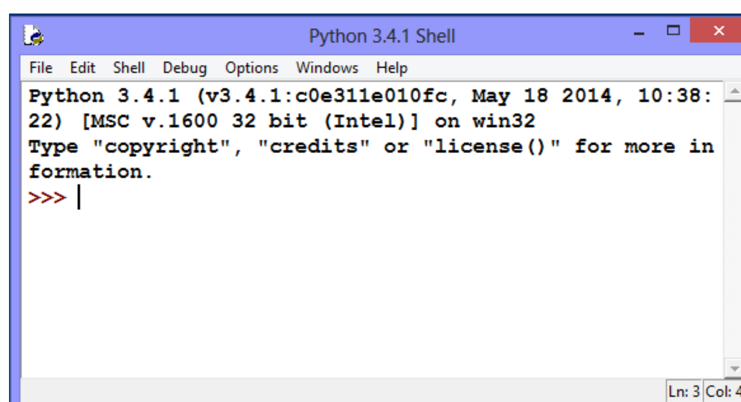
https://www.python.org/



La version más reciente disponible para WINDOWS es la **versión 3.4.1** de mayo 18, 2014

La instalación es guiada mediante pantallas con opciones que deben aceptarse. Cuando la instalación está completa se habrá creado una carpeta con el nombre **Python34**.

La instalación también crea en la barra del menú inicial de WINDOWS y en la carpeta del programa a la que se accede desde el menú inicial, un ícono con el nombre **IDLE (Python GUI)** mediante el cual se despliega una pantalla de interfaz que facilta la interacción con Python:



Esta es la pantalla principal o Shell de Python. Esta pantalla incluye en el margen superior un menú de opciones. Al presionar **file** en el menú se abren opciones para crear ventanas de edición de programas, para buscar programas, guardarlos, etc.

En este documento usaremos este dispositivo para interactuar en línea con Python y también para el ingreso y salida de los resultados cuando se escriban programas y se realice la ejecución. Aparte del IDLE que ofrece Python, existen otras instrumentaciones disponibles como interfaz de Python.

Se puede personalizar la apariencia de esta pantalla presionando el botón **Options** en el menú y seleccionando **Configure IDLE**.

IDLE proviene de las palabras Interactive Development Environment.
GUI proviene de las palabras Graphical User Interface

Nota: Si en lugar de **IDLE (Python GUI)** elije **Python (command line)** se abrirá la ventana primitiva de Python la cual es muy limitada para interactuar con el lenguaje.

5.4 Extensiones al lenguaje

Para enriquecer el lenguaje, los usuarios de la comunidad Python han desarrollado muchas extensiones organizadas en paquetes o librerías las cuales están disponibles de manera gratuita en la red internet para integrarlas al entorno del lenguaje.

Un sitio de internet que contiene una gran cantidad de paquetes con extensiones para Python ha sido compilada por Christoph Gohlke de la Universidad de California en Irvine. De este sitio se pueden descargar e instalar para WIINDOWS, en la dirección:

http://www.lfd.uci.edu/~gohlke/Pythonlibs/

Los siguientes son algunos de los paquetes de extensión importantes que se pueden descargar de este sitio para WINDOWS. Se recomienda realizar su instalación.

NumPy: Paquete fundamental para computación matemática y científica con Python
Matplotlib, Pylab: Librería para gráficos en dos y tres dimensiones
SymPy: Librería para aplicaciones con matemáticas simbólicas

Al descargar estas extensiones, el programa de instalación las agrega a la carpeta **site-packages** ubicada dentro de **lib** la cual a su vez está incluida en la carpeta **Python34**. La carpeta **site-packages** contiene algunos paquetes iniciales que se agregan directamente al instalar Python. La librería **Matplotlib** incluye a **Pylab**.

El sitio oficial para decargar las librerías fundamentales: Numpy, SciPy, SymPy, Matplotlib:

http://www.scipy.org/

Existe gran cantidad de información acerca del uso del lenguaje Python en la red internet, así como ejemplos y videos en el portal **YouTube**.

Algunos sitios de interés en la red internet con información para el aprendizaje de Python:

Sitio web con información incial para la instalación y uso de del lenguaje Python

https://wiki.python.org/moin/BeginnersGuide/

Un tutorial detallado del lenguaje Python.

http://www.tutorialspoint.com/Python/

En la siguiente dirección se puede encontrar y bajar en formato pdf el tutorial elaborado por Guido Van Rossum y traducido al español por voluntarios usuarios de Python en Argentina.

http://docs.Python.org.ar/tutorial/pdfs/TutorialPython3.pdf

En general, estos documentos proveen información dispersa acerca del uso del lenguaje, pero no incluyen muchos ejemplos de aplicación.

El documento que ofrecemos en esta obra es una integración de muchos temas de interés para el aprendizaje del lenguaje Python, con una gran cantidad de ejemplos instructivos y ejercicios de práctica para los interesados.

5.5 Desarrollo de programas en el lenguaje Python

Un programa es la descripción de un algoritmo en un lenguaje computacional. En la terminología de Python, un programa se denomina **módulo**.

Para escribir un programa es necesario haber conceptualizado o construido el algoritmo para resolver el problema de interés y conocer las reglas de sintaxis y semántica del lenguaje computacional que será usado.

También deberá tener instalado en su computador el traductor del lenguaje. En este documento se usará una interfaz de Python para desarrollo denominada IDLE (Interactive DeveLopment Environment). Adicionalmente se pueden instalar extensiones del lenguaje que han sido desarrollados para aplicaciones especiales.

5.6 Algunos elementos básicos para escribir programas

5.6.1 Tipos de datos básicos

Son los componentes elementales con los que se opera en el lenguaje Python. En color azul se muestra el nombre del tipo.

a) Enteros (int)

Son números sin punto decimal.

Ejemplos.

```
37
0
-128
```

b) Reales o números de punto flotante (float)

Números con punto decimal o expresados en notación de potencias de 10

Ejemplos.

```
3.25
-0.028
2.4e-5
```

es el número $2.4x10^{-5}$

c) Complejos (complex)

Números expresados con un componente real y un componente imaginario

Ejemplo.

```
(2+3j)
```

d) Cadenas de caracteres (str)

Expresiones encerradas entre comillas simples o comillas dobles. Este tipo de datos no es básico. Pero damos una primera mirada. Serà estudiado en detalle en otra sección.

Ejemplos.

`'un algoritmo'` o `"un algoritmo"`

e) Valores lógicos (bool)

`True`	Representa al valor lógico **verdadero**
`False`	Representa al valor lógico **falso**

5.6.2 Variables o identificadores

Son los símbolos para representar los valores y otros componentes de los programas. Para escribir variables se pueden usar letras, mayúsulas y minúsculas, dígitos y el sub-guión pero deben comenzar con una letra o con el sub-guión. Se pueden usar tildes y **ñ**.

Las variables se crean al asignarles un valor. Esta asignación se denomina dinámica pues se realiza durante la ejecución, es decir que las variables se crean y pueden modificarse durante la ejecución. El tipo de datos de la variable se define con el tipo del valor asignado.

Se recomienda que los nombres de variables no coincidan con las palabras reservadas que tienen un significado especial para el lenguaje de programación. Se sugiere usar nombres cortos pero significativos y relacionados con los valores que representarán.

Lista (no exhaustiva) de **palabras reservadas** de Python

```
and  assert  break  class  continue  def  del  elif  else  except  exec
finally  for  from  global  if  import  in  input is  lambda  next  not
or  pass  print  raise  return  try  while  yield
```

5.6.3 Operadores

Son los símbolos utilizados para expresar las operaciones básicas en los programas

a) Operadores aritméticos

Se utilizan para escribir expresiones aritméticas. También se pueden usar los paréntesis **( )** para definir el orden de las operaciones. El resultado es un valor aritmético.

Operación	Python
Suma	+
Resta	-
Multiplicación	*
División real	/
Potenciación	**

Ejemplos. $(a+2)^3$ traducción al lenguaje Python: **`(a+2)**3`**

$\frac{a+5}{b-1}$ traducción al lenguaje Python: **`(a+5)/(b-1)`**

b) Operadores relacionales

Estos símbolos se usan para comparar valores. El resultado de esta comparación es un valor lógico: `True` o `False`.

Matemáticas	Python
<	`<`
>	`>`
≤	`<=`
≥	`>=`
=	`==`
≠	`!=`

Ejemplo. `x <= 5`

El resultado será `True` si el contenido de **x** es menor o igual que **5**, caso contrario, será `False`

c) Conectores lógicos

Estos símbolos se utilizan para construir expresiones lógicas. El resultado es un valor lógico `True` o `False`.

Matemáticas	Python
Conjunción: $\wedge$	`and`
Disyunción: $\vee$	`or`
Negación: ˥	`not`

Ejemplo. `x < 5 and t > 2`

Dependiendo de **x** y **t** el resultado será `True` o `False`

d) Precedencia de operadores

Si en una expresión hay operadores de diferente tipo, primero se evalúan las operaciones **aritméticas**, luego las operaciones **relacionales** y finalmente las operaciones **lógicas.**

Los **paréntesis ( )** se pueden usar para definir con claridad la precedencia de las operaciones.

c) Operadores especiales

Existen otros operadores. Aquì incluimos dos operadores de uso frecuente

Operador de inclusión

```
e  in  c
e  not in  c
```

Este operador detecta si un elemento **e** pertenece o no a una colección de datos **c**. Las colecciones de datos serán revisadas en una sección posterior. Una cadena o string es una colección de datos. El resultado de esta operación es un valor lógico: `True` o `False`

Ejemplo:

```
>>> 'm'  in  'programa'
```

El resultado es `True`

Operador de concatenación

El operador **+** se puede usar para concatenar colecciones de datos. Una cadena o string es una colección de datos. Si se concatenan cadenas, el resultado será una cadena compuesta por ambas cadensa.

Ejemplo.

```
>>> x='Mate'
>>> y='mática'
>>> z=x+y
>>> z
'Matemática'
```

Al escribir la variable se muestra el contenido
El resultado es una cadena concatenada

```
>>> z='La '+z
>>> z
'La Matemática'
```

Puede usarse en forma recurrente

```
>>> r='e' in z
>>> r
True
```

Nota: Los colores aparecen automáticamente al escribir instrucciones en la ventana de Python

5.6.4 Conversión entre tipos de datos

Siempre que el contenido sea compatible, se puede convertir entre tipos de datos mediante una especificación correspondiente al tipo de datos requerido.

Ejemplos.

Asignación	Resultado	Tipo del resultado
`a=73`	`73`	Entero
`b=float(a)`	`73.0`	Real
`c=str(a)`	`'73'`	Cadena
`d=73.5`	`73.5`	Real
`e=int(d)`	`73`	Entero
`s='125'`	`'125'`	Cadena
`g=int(s)`	`125`	Entero
`s='125.7'`	`'125.7'`	Cadena
`r=float(s)`	`125.7`	Real
`t=int(s)`		Error de conversión
`s='n728'`	`'n728'`	Cadena
`t=int(s)`		Error de conversión
`x=5`	`5`	Entero
`z=complex(x)`	`(5+0j)`	Complejo
`u=2+3j`	`(2+3j)`	Complejo
`a=u.real`	`2.0`	Real (Componente real de u)
`b=u.imag`	`3.0`	Real (Componente imaginario de u)
`y=float(u)`		Error de conversión
`s='2+3j'`	`'2+3j'`	Cadena
`z=complex(s)`	`(2+3j)`	Complejo
`t=2<3`	`True`	Lógico
`a=75`	`75`	Entero
`b=chr(a)`	`'K'`	Carácter cuyo código entero (ASCII) es **75**
`c='R'`	`'R'`	Carácter
`d=ord(c)`	`82`	Código entero que representa el carácter **'R'**

Los colores en los ejemplos son para resaltar y distinguir los nombres de Python. Estos colores aparecen automáticamente al escribir expresiones en la ventana de Python.

5.6.5 Tipos numéricos en otras bases

Números en binario

Base: **2**
Símbolos: **0,1**
Formato: **0bdddd.....** (dígitos en binario)
Ejemplo: **0b1001101101**

Números en octal

Base: **8**
Símbolos: **0,1,2,3,4,5,6,7**
Formato: **0odddd.....** (dígitos en octal)
Ejemplo: **0o2536174**

Números en hexadecimal

Base: **16**
Símbolos: **0,1,2,3,4,5,6,7,8,9,a,b,c,d**
Formato: **0xdddd.....** (dígitos en hexadecimal)
Ejemplo: **0x36a7d0a9**

Se puede operar con números de diferente base. Python muestra el resultado en formato decimal.

Ejemplo. Sumar números con diferente base

```
>>> 7352 + 0o123 + 0x2ac4 + 0b10011
18402
```

Se puede convertir de decimal a otra base con las funciones de conversión de tipo:

Asignación	Resultado	Tipo del resultado
n=93		
q=bin(n)	'0b1011101'	Cadena con la representación en binario de 93
r=oct(n)	'0o135'	Cadena con la representación octal de 93
s=hex(n)	'0x5d'	Cadena con la representación hexadecimal de 93

La conversión de binario, octal o hexadecimal a decimal es automática. No se requiere aplicar la función de conversión de tipo:

Ejemplo.

```
>>> u=0b1011101
>>> u
93

>>> u=int(0b1011101)          La conversión de tipo no es necesaria
>>> u
93
```

5.6.6 Uso de módulos especiales

Los algoritmos escritos en el lenguaje Python se denominan programas o módulos. Estos módulos se almacenan con algún nombre en alguna carpeta en el disco. También existen otros módulos especiales o **librerías** que deben cargarse para tener acceso a estos recursos.

Algunos de estos módulos especiales se instalan desde el inicio en la librería estándar del traductor Python. Otros se los puede descargar de la red internet y otros pueden ser creados por los propios usuarios.

Python incluye los operadores e instrucciones básicas pero si se necesitan funciones especiales se debe cargar el módulo o librería que las contiene con la siguiente sintaxis:

```
from módulo import función
```

Las funciones matemáticas comunes están en el módulo **math**.

Ejemplo. Si se desea usar la función **coseno**, puede escribir:

```
from math import cos
```

Para cargar **todas** las funciones del módulo **math** debe especificarse con un asterisco:

```
from math import*
```

Ejemplo. Asignar a **x** el logaritmo natural de $\sqrt{3}$

```
x=log(sqrt(3))
```

Otro formato para importar un módulo usa la siguiente especificación:

```
import módulo
```

En este caso las funciones incluídas en el módulo deben referenciarse con la notación:

```
módulo.función
```

Ejemplo. Importar el módulo **math**

```
import math
```

Asignar a **x** el logaritmo de $\sqrt{3}$

```
x=math.log(math.sqrt(3))
```

Se puede importar un módulo para usarlo con otro nombre

Ejemplo. Importar el módulo **math** para usarlo con el nombre **mt**

```
import math as mt
```

Asignar a **x** el logaritmo de $\sqrt{3}$

```
x=mt.log(mt.sqrt(3))
```

Un módulo para acceder al reloj**: time**

Ejemplo. Mostrar la fecha y hora actual formateadas

```
from time import*
asctime()
'Fri Jul 12 15:38:25 2014'
```

Ejemplo. Mostrar el tiempo real de ejecución de un proceso

```
from time import*
clock()
4.6651125621684564e-07
. . .
clock()
3.9174629625050907
```

5.6.7 El sistema de ayuda

Python incluye un sistema de ayuda en línea con la sintaxis:

```
help('item')
```

En donde **'item'** representa el tema para el que se solicita ayuda:

Ejemplo. Si desea conocer el nombre y uso de todas las funciones matemáticas del módulo **math** escriba:

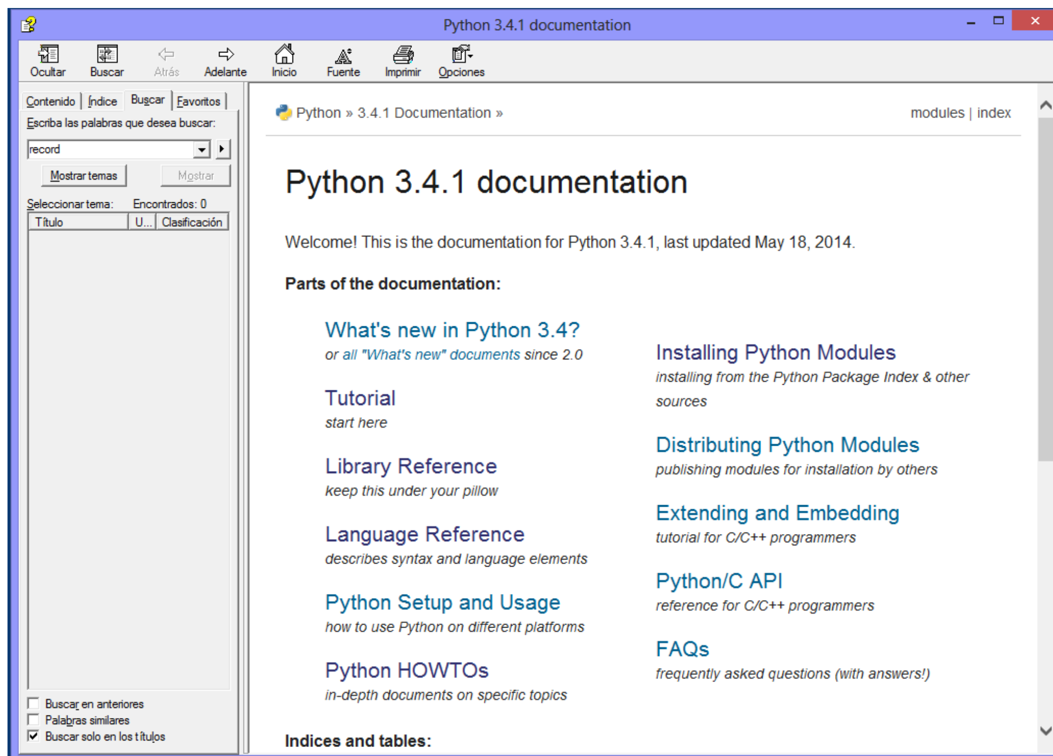
```
help('math')
```

Ejemplo. Si desea conocer información del tipo de datos **int** escriba:

```
help('int')
```

5.6.8 Documentación en linea

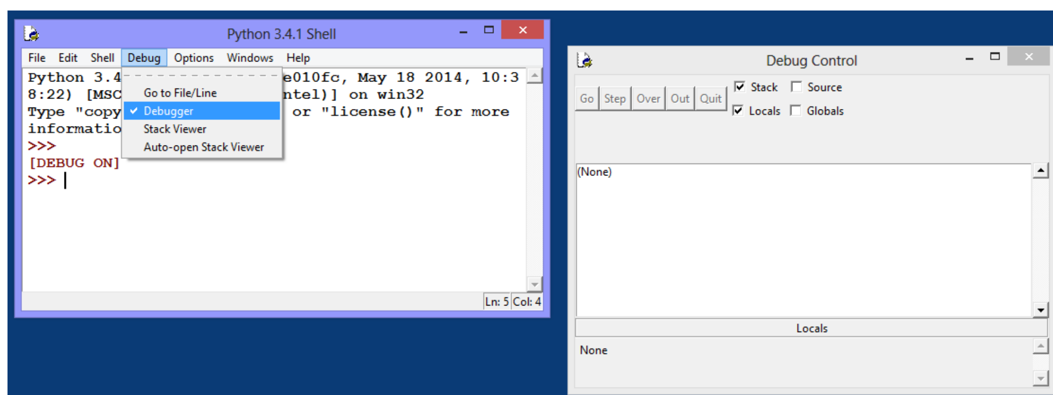
Desde la ventana de Python, si presiona la tecla funcional **F1** se tiene acceso a la documentación de Python incorporada en el Shell, incluyendo las características de la versión instalada, tutorial de uso del lenguaje, la información fundamental de la librería estándar de Python, el manual de referencia del Lenguaje Python, etc. como aparece en la pantalla:



Siguiendo las opciones que ofrecen el documento se puede llegar a un nivel de detalle fino

5.6.9 Depuración de programas

Python dispone de un dispositivo de seguimiento y depuración de programas. Para acceder a esta pantalla selecciones **Debugger** de la opción **Debug** de la barra del menú.



El uso de estas opciones es útil para usuarios que enfrentan problemas complicados en el desarrollo de programas complejos.

5.6.10 Funciones del módulo math

Este módulo provee acceso a las funciones matemáticas comunes:

Las funciones se muestran en el siguiente cuadro en orden alfabético:

Nombre	Resultado
`abs(x)`	Valor absoluto
`acos(x)`	Arco coseno de x en radianes
`acosh(x)`	Arco seno hiperbólico
`asin(x)`	Arco seno
`asinh(x)`	Arco seno hiperbólico
`atan(x)`	Arco tangente
`atanh(x)`	Arco tangente hiperbólico
`ceil(x)`	Entero menor
`cos(x)`	Coseno trigonométrico de x en radianes
`cosh(x)`	Coseno hiperbólico
`degrees(x)`	Conversión de radianes a grados
`erf(x)`	Función error
`exp(x)`	Función exponencial
`fabs(x)`	Valor absoluto de un real
`factorial(x)`	Factorial de un entero positrivo
`floor(x)`	Entero mayor
`gamma(x)`	Función Gamma
`hypot(x,y)`	Distancia Euclideana
`log(x)`	Logaritmo natural
`log(x,b)`	Logaritmo de x, en base b
`log10(x)`	Logaritmo base 10
`log2(x)`	Logaritmo en base 2
`modf(x)`	Parte decimal y parte entera de x
`pow(x,y)`	x elevado a la potencia y
`radians(x)`	Conversión de radianes a grados
`sin(x)`	Seno trigonométrico de x en radianes
`sinh(x)`	Seno hiperbólico
`sqrt(x)`	Raíz cuadrada
`tan(x)`	Tangente trigonométrica de x en radianes
`trunc(x)`	Truncamiento de decimales hacia 0
`e`	Constante **e** = 2.718281828459045
`pi`	Constante π = 3.141592653589793

5.6.11 Traducción de expresiones

En esta sección se realiza una práctica de escritura de expresiones en la notación Python

Ejemplo. Traduzca al lenguaje Python la expresión aritmética:

$$\frac{3^{0.75}\sqrt{2}}{e^2 - 1}$$

Traducción:

```
from math import*
3**0.75*sqrt(2)/(exp(2)-1)
```

Las funciones matemáticas están **math**
El resultado será un número real

Ejemplo. Traduzca al lenguaje Python la expresión lógica:

$$a \le 2 \wedge b \ne 3$$

Traducción:

```
a<=2 and b!=3
```

El resultado será un valor lógico (`True` o `False`)

5.6.12 Ejercicios de traducción de expresiones

Traduzca al lenguaje Python cada expresión.

1) $\tan(x^3)$

2) $\dfrac{\cos(\pi + 0.2) - 3}{\tan(\sqrt{2} - 2) + 0.2^3}$

3) $\dfrac{\dfrac{\sqrt{2}+1}{\text{sen}(2) - 0.1}}{x^2 - 1} + 2$

4) $\rceil(b < 5 \vee c = 0)$

5) $(a < b) \Rightarrow (c = 5)$ Sugerencia: Use la equivalencia lógica $\mathbf{p \Rightarrow q \equiv \rceil p \vee q}$

5.6.13 Un ejemplo introductorio

Para iniciar el aprendizaje del lenguaje Python en esta sección se desarrolla un ejemplo en **forma interactiva** en la ventana principal o shell. Se recomienda que el usuario realice esta práctica en la computadora.

El mismo ejemplo se lo resolverá posteriormente escribiendo un programa en una ventana de edición de Python. Esta práctica permitirá resaltar las diferencias entre el modo interactivo y el modo de programación.

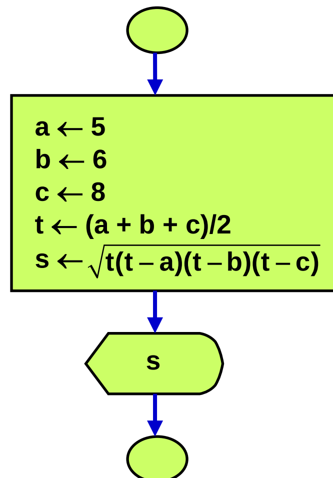
Ejemplo. Describa un algoritmo para calcular el área de un triángulo cuyos lados son:
5, 6, y **8**

Algoritmo: Área de un triángulo conocidos sus lados

Variables

a, b, c: Lados del triángulo (Datos conocidos: **5**, **6**, y **8**)
s: Área del triángulo (Es el resultado esperado)
t: Semiperímetro (Valor usado para la fórmula del área
$s = \sqrt{t(t-a)(t-b)(t-c)}$, (Fórmula del área del triángulo)
siendo **t = (a + b + c)/2**

Diagrama de flujo



Solución en modo interactivo en la ventana principal o shell de Python

Presione el ícono de Python e ingrese a la ventana principal o Shell. En el borde superior se muestra un menú de opciones. También se despliega alguna información de la versión del programa que está siendo utilizado. Al inicio de las siguientes líneas se muestra un aviso para escribir cada instrucción. Este aviso son tres ángulos: **>>>**

En esta ventana escriba cada instrucción a la derecha del símbolo **>>>** Al final de cada línea presione la tecla de ingreso. Siga el ejemplo que se muestra en el gráfico a continuación.

En esta ventana se ingresan las instrucciones las cuales son interpretadas y ejecutadas inmediatamente en forma parecida a una calculadora. Para conocer el contenido de las variables se puede escribir el nombre de la variable.

```
Python 3.4.1 Shell
File  Edit  Shell  Debug  Options  Windows  Help
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:3
8:22) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more
information.
>>> from math import*
>>> a=5
>>> b=6
>>> c=8
>>> t=(a+b+c)/2
>>> s=sqrt(t*(t-a)*(t-b)*(t-c))
>>> s
14.981238266578634
>>>
Ln: 11 Col: 4
```

La figura anterior es tomada de la interacción real con el lenguaje Python. Los colores son asignados automáticament al escribir las instrucciones pero pueden personalizarse, igualmente el tipo y tamaño de las letras, el tamaño inicial de la ventana, la tabulación, etc. Puede hacerlo seleccionando la opción **Configure IDLE** de **Options** en el menú de la ventana principal.

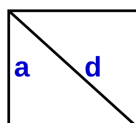
5.6.14 Práctica computacional en la ventana interactiva

Para afirmar el conocimiento adquirido se realizará una práctica en la pantalla interactiva de Python resolviendo problemas básicos. Esta es la pantalla principal o shell.

Ejemplo. Resuelva el siguiente ejercicio en la ventana interactiva de Python

Si se conoce que el área de un cuadrado es 40 m^2, encuentre el valor de la diagonal

Formulación



a: Longitud del lado del cuadrado
d: Longitud de la diagonal

$a^2 = 40 \Rightarrow a = \sqrt{40}$ (Dato del área)

$d^2 = 2\,a^2 \Rightarrow d = a\sqrt{2}$ (Teorema de Pitágoras)

En la ventana principal de Python se escriben las instrucciones respectivas:

```
>>> from math import*
>>> a=sqrt(40)
>>> d=a*sqrt(2)
>>> d
8.94427190999916
```

Al escribir la variable **d** se muestra el resultado calculado.

5.6.15 Ejercicios de resolución de problemas en la ventana interactiva

Para cada ejercicio escriba la formulación necesaria. Después escriba las instrucciones en la ventana principal o shell de Python y obtenga la respuesta.

1) Calcule el área total de un bloque de dimensiones 20, 30, 40 cm

2) Calcule el área total de un cilindro de radio 5 y altura 4 metros

3) Calcule el área de un triángulo rectángulo cuyos diagonal mide 5 cm. y tiene un ángulo interno de 40 grados.

4) El costo mensual **c** en dólares al fabricar una cantidad **x** de artículos está dado por: **c = 50 + 2x**, mientras que el ingreso por la venta de **x** artículos está dada por: **v = 2.4x**

a) Calcule la ganancia que se obtendrá al fabricar y vender 400 artículos

b) Determine cuantos artículos deben fabricarse y venderse para que el ingreso iguale a los gastos

5) Un modelo de crecimiento poblacional está dado por $f(t) = kt + 2e^{0.1t}$, siendo **k** una constante que debe determinarse y **t** es tiempo en años. Se conoce que f(10)=50. Determine la población en el año 25

6) Un ingeniero desea tener una cantidad de dólares acumulada en su cuenta de ahorros para su retiro luego de una cantidad de años de trabajo. Para este objetivo planea depositar un valor mensualmente. Suponga que el banco acumula el capital mensualmente mediante la siguiente fórmula:

$$a = p\left[\frac{(1+x)^n - 1}{x}\right]$$

a: valor acumulado luego de **n** depósitos mensuales
p: valor de cada depósito mensual
x: valor nominal del interés mensual
n: número de depósitos mensuales realizados

a) Calcule el valor acumulado en 15 años depositando mensualmente cuotas de 300 con un interés **anual** de 0.04

b) Determine el valor de la cuota que debe depositar mensualmente si desea reunir 200000 en 20 años suponiendo que el interés **anual** es 0.04

c) Determine cuantos depósitos mensuales de 400 debe realizar para reunir 250000 con una tasa de interés **anual** de 0.04

5.7 Instrucciones básicas para programar con Python

El modo interactivo de Python facilita la obtención inmediata de respuestas, pero si se requiere resolver un problema similar pero con datos diferentes, se tendrían que digitar nuevamente las instrucciones. Es preferible que las instrucciones estén almacenadas en un archivo con algún nombre. Estos archivos son los programas o **módulos** que se ejecutan para obtener la respuesta de interés.

Los detalles de la sintaxis del lenguaje Python se los revisará en las siguientes secciones.

Cada instrucción básica del lenguaje Python se describirá relacionándola con su equivalente en la notación algorítmica desarrollada en un capítulo anterior en este documento.

5.7.1 Instrucción de asignación

Esta instrucción se usa para definir variables y asignar un valor a su contenido

Sean **v** una variable y **r** un valor

Notación algorítmica

v ← r   Asigna el valor **r** a la variable **v**

Seudo lenguaje

v ← r

Lenguaje Python

```
v=r
```

Ejemplo. Asignaciones en el lenguaje de Python

```
x=3
y=2*x+1
```

5.7.2 Asignaciones especiales

a) Asignaciones en la misma línea. Deben separarse con punto y coma. Las asignaciones se realizan de izquierda a derecha.

Ejemplo.

```
x=3; t=x+2; n=4
```

b) Asignación múltiple. Deben separarse con comas. La asignación se realiza en forma correspondiente, a cada variable, se asigna cada valor de izquierda a derecha.

Ejemplo.

```
a,b,c=5,7,'saludo'
```

Es equivalente a

```
a=5
b=7
c='saludo'
```

c) Asignación a varias variables el mismo valor

Ejemplo.

```
a=b=c=d=0
```

Es equivalente a

```
a=0
b=0
c=0
d=0
```

d) Intercambio del contenido de dos variables

Ejemplo.

```
a=3
b=5
a,b=b,a                          (a contendrá 5 y b contendrá 3)
```

Es equivalente a

```
a=3
b=5
x=a
a=b
b=x
```

e) Notación abreviada para operar y asignar

Ejemplo.

```
a=a+b
```

Se puede escribir en forma abreviada

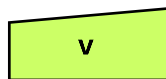
```
a+=b
```

También es aplicable a las otras operaciones aritméticas

5.7.3 Instrucción para ingreso de datos

Esta instrucción se usa para describir la acción de ingresar algún valor para una variable desde fuera del algoritmo cuando éste sea ejecutado. Esta instrucción permite que los datos no requieran ser asignados dentro del algoritmo y así puedan realizarse pruebas con diferentes datos sin tener que modificar las instrucciones del programa.

Notación algorítmica



Ingresar un valor para la variable **v** desde el teclado

Seudo lenguaje

Entrar v

Lenguaje Python

```
v=input('mensaje ')
```

Esta instrucción puede incluir un mensaje que se mostrará al ejecutar el programa en la ventana principal o shell para indicar que es el momento de ingresar el dato. Este mensaje puede escribirse entre comillas simples o entre comillas dobles.

Python recibe el valor ingresado como un dato de **tipo texto**. Si se desea asignar otro tipo de dato a este valor, es necesario aplicar una conversión de tipo:

Para convertir el **texto recibido** a un valor entero:

```
v=int(input('mensaje '))
```

Para convertir el **texto recibido** a un valor decimal (o de punto flotante):

```
v=float(input('mensaje '))
```

Ejemplos.

Ingresar un dato tipo texto

```
s=input('Ingrese su nombre: ')
```

Ingrese un dato (número entero) y conviértalo a tipo numérico entero

```
n=int(input('Ingrese la cantidad de hijos: '))
```

Ingrese un dato (número entero o real) y conviértalo al tipo numérico real

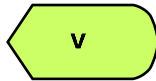
```
p=float(input('Ingrese su peso en kg.: '))
```

Si la conversión no puede hacerse, se producirá un error de conversión de tipo.

5.7.4 Instrucción para salida de resultados

Esta instrucción se usa para describir la acción de mostrar mensajes o el contenido de resultados almacenados en variables. Para mostrar el contenido de una variable se debe escribir el nombre de la variable.

Notación algorítmica



Mostrar en la pantalla el contenido de la variable **v**

Seudo lenguaje

Mostrar v

Lenguaje Python

```
print(v)
```

Se pueden mostrar mensajes entre comillas simples o dobles junto al contenido de una o más variables separadas con comas. La salida se despliega en la ventana principal o shell de Python. En las primeras versiones de Python el uso de los paréntesis era opcional.

Ejemplos. En los ejemplos se supondrá que las variables contienen algún valor

```
print(x)
print('El resultado es: ', x )
print('El area es: ', s, ' El volumen es: ', t )
```

Se pueden incluir algunos caracteres de control de salida. Si se desea un salto a la siguiente línea en pantalla se deberá escribir `'\n'`

Ejemplo.

```
print('El area es: ', s, ' \nEl volumen es: ', t )
```

Especificaciones de formato para salida

Opcionalmente, la instrucción **print** puede incluir especificaciones de formato para mejorar la apariencia de los resultados que se muestran en la pantalla. Una forma conocida de estas especificaciones requiere escribirlas entre comillas precedidas con el símbolo **%**

Algunas especificaciones de formato se escriben: **%cd**, **%c.pf**, **%cs**, **%c.pg**
En donde **c, p** son el número de columnas y número de decimales o dígitos respectivamente, mientras que **d, f, s, g** se refieren en ese orden a datos de tipo entero(decimal), real(flotante), cadena(string) y en notación de potencias de 10.

Ejemplos.

```
n=23
x=7.78925
e='ESPOL'
u=4.5**12
print('Prueba %5d%8.2f%10s%12.5g'%(n,x,e,u))
Prueba    23    7.79     ESPOL   6.8953e+07
```
resultados formateados

Las especificaciones de formato se pueden almacenar con un nombre:

```
f='Prueba %5d%8.2f%10s%12.5g'
print(f%(n,x,e,u))
```

También se puede especificar el formato con la palabra especial **format**

```
print(format(n,'5d'))
   23
print(format(x,'8.2f'))
    7.79
```

5.7.5 Documentación de los programas

Es una buena práctica de programación incluir **comentarios** en los programas para documentar su desarrollo.

Para incluir comentarios o anotaciones en el programa inicie la línea con el símbolo **#**

Los comentarios también pueden abarcar varias líneas. Para estas anotaciones escriba al inicio de la primera línea del comentario **tres comillas simples o dobles** y también escríbalas al final de la última linea del comentario.

```
# Esta es una línea de comentario

'''
Estas líneas
son comentarios
'''
```

Se pueden usar líneas en blanco para mejorar la claridad de los programas.

5.7.6 Encolumnamiento de instrucciones

En el lenguaje Python se definen los bloques encolumnando las instrucciones a la derecha debajo de la instrucción de control en cuyo ámbito estarán incluidas. A diferencia de otros lenguajes, Python no tiene símbolos o palabras especiales para delimitar un bloque.

Las instrucciones que pertenecen a un bloque deben escribirse desplazadas al menos una columna a la derecha dentro del bloque y todas las instrucciones deben tener el mismo encolumnamiento. Python sugiere el encolumnamiento al momento de escribir las instrucciones del programa. Es importante constatar el encolumnamiento de las instrucciones para asegurar su pertenencia o exclusión de un bloque.

El encolumnamiento de las instrucciones para definir el ámbito o alcance de las estructuras de control es muy simple de aplicar y entender, aporta claridad a la escritura de los programas y es un componente importante de la metodología de la **Programación Estructurada**.

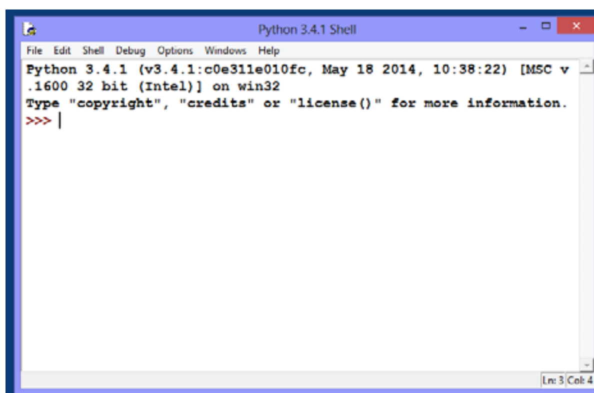
5.7.7 El primer programa

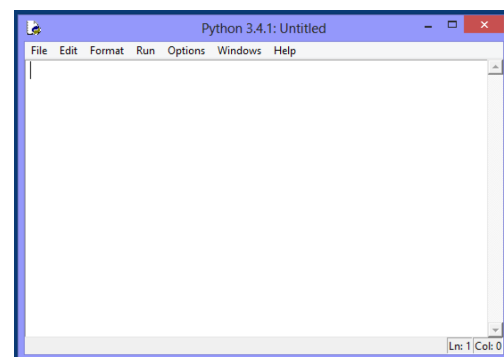
Cuando un usuario enfrenta a un nuevo lenguaje, es tradicional que su primer programa sea el clásico **“Hola mundo”**. Este programa consiste en hacer que el computador muestre en la pantalla un saludo. En lenguajes “duros” como C++ o Java, esta simple actividad involucra varios detalles que para un usuario novicio es difícil entender. En Python escribir y probar programas es una actividad simple y amigable

El programa “Hola mundo” en Python

Primero ingrese a la ventana principal o Shell de Python activando el programa. En esta ventana presione **file** en la barra del menú y elija la opción **New File.** Se abrirá una ventana de edición.

Ventana interactiva o Shell | **Ventana de edición**

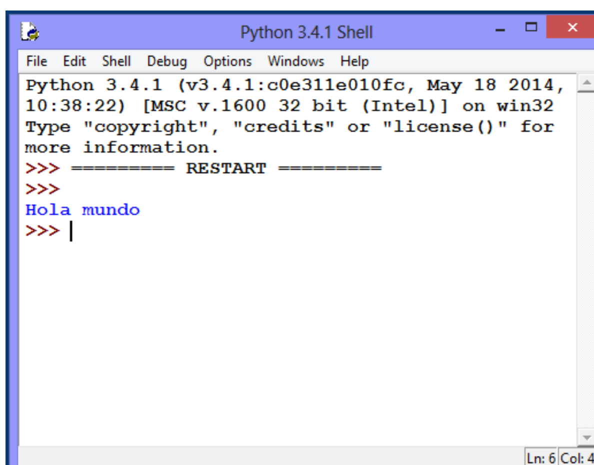


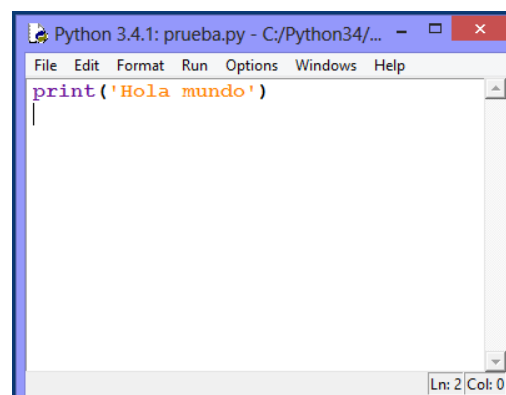


Escriba en la ventana de edición de Python la instrucción

```
print('Hola mundo')
```

Almacene el contenido de la ventana de edición. Este contenido es su primer programa. Para almacenar el programa presione **file** en la barra del menú de la ventana de edición y elija la opción **save**. Python le pedirá un nombre. Escriba algún nombre para su programa sin dejar espacios intermedios Para probar o ejecutar el programa elija la opción **run** de la ventana de edición o presione la tecla **F5**. El siguiente gráfico muestra el resultado que aparece en la pantalla interactiva.





El segundo ejemplo será convertir en un programa el ejemplo del triángulo que se desarrolló interactivamente en la ventana interactiva de Python. Los datos serán ingresados desde el teclado para realizar varias pruebas. Primero se codificará el programa con las reglas indicadas anteriormente y luego se lo trasladará a la ventana de edición. El ejemplo permitirá entender la ventaja de escribir programas.

Ejemplo. Escriba en el lenguaje Python un programa para traducir el algoritmo que calcula el área de un triángulo ingresando el valor de sus tres lados

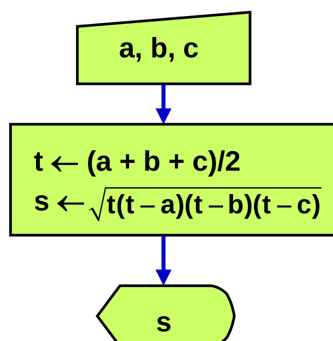
Algoritmo: Área de un triángulo dados sus lados

Variables

a, b, c:	Lados del triángulo	(Datos desconocidos)
s:	Área del triángulo	(Es el resultado esperado)
t:	Semiperímetro	(Valor usado para la fórmula del área
$s = \sqrt{t(t-a)(t-b)(t-c)}$,		(Fórmula del área del triángulo)

siendo **t = (a + b + c)/2**

Diagrama de flujo



Traducción del algoritmo al lenguaje Python

```
#Programa calcular el área de un triángulo
from math import sqrt
a=float(input('Primer lado: '))
b=float(input ('Segundo lado: '))
c=float(input ('Tercer lado: '))
t=(a+b+c)/2
s=sqrt(t*(t-a)*(t-b)*(t-c))
print('Respuesta: ',s)
```

Al escribir las instrucciones del algoritmo en una página, no es necesario usar colores como se muestran en el cuadro anterior. Pero al trasladar el programa a una ventana de Python los colores aparecen automáticamente al escribir las instrucciones.

Estos colores pueden personalizarse, igualmente el tipo y tamaño de las letras, el tamaño inicial de la ventana, la tabulación, etc. Puede hacerlo seleccionando la opción **Configure IDLE** de **Options** en el menú de la ventana principal. Se sugiere no modificar ni el tipo de letra ni la tabulación.

Escriba el programa en la ventana de edición de Python. Elija la opción **save** o **save as** y escriba un nombre para su programa. Python agrega la extensión **.py** al nombre.

Para probar o ejecutar el programa elija la opción **run** de la ventana de edición en la que escribió el programa, o presione la tecla **F5**. Esta forma de ejecutar programas será usada en casi todos los ejemplos en este documento.

El ingreso de los datos y la salida de resultados se realiza en la ventana principal o Shell como se muestra en la siguiente figura tomada de la interacción real con Python.

Ventana de principal o Shell **Ventana de edición con el programa**

Python 3.4.1 Shell

File Edit Shell Debug Options Windows Help

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:3
8:22) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more
information.
>>> ================================ RESTART =======
========================
>>>
Primer lado: 4.5
Segundo lado: 7
Tercer lado: 6.2
Respuesta:  13.738049124602805
>>> |
```

Ln: 9 Col: 4

En la ventana principal o shell se realiza la ejecución del programa

*Python 3.4.1: primo.py - C:/Users/Luis Rodriguez...

File Edit Format Run Options Windows Help

```
#Programa calcular el área de un triángulo
from math import sqrt
a=float(input('Primer lado: '))
b=float(input('Segundo lado: '))
c=float(input('Tercer lado: '))
t=(a+b+c)/2
s=sqrt(t*(t-a)*(t-b)*(t-c))
print('Respuesta: ',s)
```

Ln: 9 Col: 0

En la ventana de edición se escribe el programa y se ordena su ejecución

La ventaja de un programa es su independencia con respecto a los datos. Los datos entran desde fuera del programa cuando es ejecutado. De esta manera el programa no necesita modificarse para realizar pruebas. Al estar almacenado se lo puede cargar y ejecutar en cualquier momento y no necesita escribir las instrucciones nuevamente.

Para realizar cambios en el programa que está abierto en la ventana de edición, posicione el cursor en el lugar respectivo, realice los cambios, almacene nuevamente el archivo y ejecútelo. Puede cargar y modificar los programas que están almacenados.

Los módulos (programas y funciones) son almacenados en una carpeta identificada con el nombre **idlelib** ubicada dentro de **lib** en la carpeta **Python34**. Python usa esta carpeta como dispositivo de almacenamiento normal de programas y archivos. Si los módulos se almacenan en otra carpeta, no se los puede acceder en forma directa y no aparecen en el directorio de módulos, aunque Python si los incluye en la lista de archivos recientes de donde se los puede cargar y ejecutar.

También se puede ejecutar desde la ventana interactiva un **programa** que está almacenado en la carpeta **idlelib**. No es necesario abrir el programa en la ventana de edición, solamente debe escribir la siguiente instrucción en la ventana interactiva. Antes de cada ejecución debe activar **Restart Shell** de la opción **Shell** del menú de Python.

```
>>> import n
```

Sustituya **n** por el nombre del programa almacenado

5.7.8 Ejercicios de programación con las instrucciones básicas

Para cada ejercicio escriba y pruebe un programa en la ventana de edición de Python.

1. Dados el radio y altura de un cilindro calcule el área total y el volumen

2. Se tiene un recipiente cilíndrico con capacidad en litros. Su altura es un dato en metros. Determine el diámetro de la base

3. Dadas las tres dimensiones de un bloque rectangular calcule y muestre su área total y su volumen.

4. La siguiente fórmula proporciona el enésimo término **u** de una progresión aritmética:

u = a + (n − 1) r

en donde **a** es el primer término, **n** es el la cantidad de términos y **r** es la razón entre dos términos consecutivos. Calcular el valor de **r** dados **u, a, n**

5. En el ejercicio anterior, calcular el valor de: **n** dados **u, a, r**

6. El examen de una materia es el 70% de la nota total. Las lecciones constituyen el 20% y las tareas el 10% de la nota total. Ingrese como datos la nota del examen calificado sobre 100 puntos, la nota de una lección calificada sobre 10 puntos, y las notas de tres tareas calificadas cada una sobre 10 puntos. Calcule la calificación total sobre 100 puntos.

7. Un modelo de crecimiento poblacional está dado por: $n = 5t + 2e^{0.1t}$, en donde n es el número de habitantes, t es tiempo en años. Se desea conocer el número de habitantes que habrían en los años 5, 10 y 20. Obtenga los resultados ejecutando tres veces el programa.

8. Un ingeniero desea tener una cantidad de dólares acumulada en su cuenta de ahorros para su retiro luego de una cantidad de años de trabajo. Para este objetivo planea depositar un valor mensualmente. Suponga que el banco acumula el capital mensualmente mediante la siguiente fórmula:

$$A = P\left[\frac{(1+x)^n - 1}{x}\right],$$ en donde

A: Valor acumulado
P: Valor de cada depósito mensual
n: Cantidad de depósitos mensuales
x: Tasa de interés mensual

Calcule el valor acumulado ingresando como datos valores para **P**, **n**, **x**

9. Lea la abscisa y ordenada de dos puntos P, Q en el plano: **(a, b)**, **(c, d)**.
Estos puntos y el origen **(0, 0)** conforman un triángulo.
Calcule y encuentre el área del triángulo.
Fórmula de la distancia del punto P al punto Q: $\mathbf{x = \sqrt{(c-a)^2 + (d-b)^2}}$
Fórmula del área del triángulo: $\mathbf{S = \sqrt{t(t-x)(t-y)(t-z)}}$, $\mathbf{t = (x+y+z)/2}$
x, y, z representan el valor de los tres lados del triángulo

5.7.9 Operadores para aritmética entera

Algunas aplicaciones numéricas incluyen operaciones aritméticas que requieren el cociente entero de la división entre dos números y el resíduo de esta división. En el lenguaje Python estas operaciones están definidas con símbolos especiales

Los operadores // y %

El operador **//** trunca los decimales del resultado de la división y entrega la parte entera.

Ejemplo.

```
>>> c=20//3
>>> c
6
```

El operador **%** entrega el módulo aritmético (residuo de la división entre dos enteros).

Ejemplo.

```
>>> r=20%6
>>> r
2
```

Ejemplo. Dado un número entero de dos cifras, escriba un programa en Python para sumar las cifras.

Solución

Variables

- **n**: dato entero de dos cifras
- **d**: dígito de las decenas
- **u**: dígito de las unidades
- **s**: suma de dígitos

Programa

```
#Suma de dos cifras
n=int(input('Ingrese un entero: ‘))
d=n//10
u=n%10
s=d+u
print('Respuesta: ',s)
```

```
Prueba del programa
>>>
Ingrese un entero: 73
Respuesta:  10
```

La función divmod

Sean **x, y** dos números

divmod(x,y) entrega el par **(x//y, x%y)**

Ejemplo.

```
>>> [a,b]=divmod(20,6)
>>> a
3
>>> b
2
```

5.7.10 Ejercicios de programación con los operadores para aritmética entera

Para cada ejercicio escriba y pruebe un programa en la ventana de edición de Python.

1. Dado un número entero (dias), determine y muestre el equivalente en años, meses y días sobrantes. Por simplicidad suponga que un año tiene 365 días y que cada mes tiene 30 días. Use los operadores **//** y **%** para obtener cociente y resíduo**.**

Ejemplo. **1372** días equivalen a **3** años, **9** meses y **7** días.

2. Dado un dato con la cantidad de días. Encuentre el equivalente en meses, semanas y días sobrantes. Suponga que cada mes tiene treinta días.

Ejemplo. Si el dato es 175 el resultado será 5 meses, 3 semanas y 4 días

3. Lea dos números de tres cifras cada uno. Sume la cifra central del primer número con la cifra central del segundo número y muestre el resultado.

4. Dado un número entero de tres cifras. Muestre el mismo número pero con las cifras en orden opuesto.

5. Dado un número entero (cantidad de dólares), mostrar el valor equivalente usando la menor cantidad de billetes de **100, 50, 20, 10, 5** y **1**.

6. En un ejercicio de algoritmos con ciclos en este documento se encuentra la siguiente fórmula para determinar el día **n** en el cual el número de bacterias **x,** cuya cantidad se duplica cada día**,** excede a un valor máximo **m:** $2^n(x) > m$.

Ingrese los valores de **x** y **m** y determine el día **n** con la fórmula anterior. Este resultado debe ser el menor entero que satisface la desigualdad. Para depejar **n** debe usar logaritmos y también truncamiento de decimales

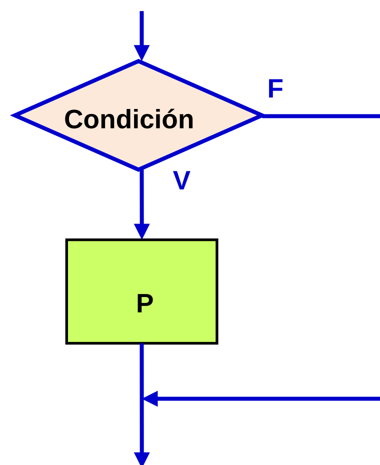
5.8 Decisiones en Python

Las decisiones son operaciones computacionales que permiten condicionar la ejecución de instrucciones dependiendo de una expresión que al evaluarla puede tomar únicamente los valores logicos: verdadero o falso, `True` o `False` en la notación de Python.

5.8.1 Ejecución condicionada de un bloque de instrucciones

Esta estructura de control se usa para condicionar la ejecución de un bloque de instrucciones utilizando como criterio el resultado de una condición como se muestra en la siguiente representación gráfica:

Representación gráfica



Al entrar a esta estructura se evalúa la condición. Si el resultado es verdadero (**V**) se ejecutarán las instrucciones del bloque **P** caso contrario, si el resultado es falso (**F**) las instrucciones del bloque no serán ejecutadas. En ambos casos el algoritmo continua abajo del bloque.

Seudo lenguaje

Si condición
        P
Fin

Lenguaje Python

```
if  condición:
     Instrucción en el bloque P
     Instrucción en el bloque P
            . . .
     Instrucción en el bloque P
```

Observe la relación entre la estructura de control del lenguaje algorítmico y la instrucción del lenguaje computacional. Ambas describen la misma acción. El color de la palabra clave **if** para decisiones de Python es el color estándar del IDLE (interfaz para interactuar con Python), pero se lo puede cambiar activando **options**.

El bloque de la estructura de decisión incluye todas las instrucciones que están condicionadas. En el diagrama de flujo está claramente delimitado con las líneas de flujo.

El lenguaje Python no tiene símbolos especiales para cerrar el bloque de instrucciones condicionadas.

El bloque o ámbito de la decisión se define encolumnando las instrucciones que se desean condicionar. Las instrucciones condicionadas deben escribirse en las siguientes líneas desplazadas al menos una columna a la derecha de la palabra **if**.

Todas las instrucciones condicionadas deben tener el mismo encolumnamiento. Python encolumna las instrucciones al escribirlas. Este encolumnamiento es una tabulación predefinida que desplaza las instrucciones cuatro espacios a la derecha y es el que se usa en este documento. Sin embargo, Python permite cambiar esta tabulación

Ejemplo. Describa en Python las instrucciones para leer el precio **p** de un producto y reducir su valor en el 10% si el dato ingresado es mayor a **40**.

```
p=float(input('Ingrese el precio: '))
if  p>40:
     p=p-0.1*p
```

Para escribir las expresiones que condicionan el bloque de instrucciones se pueden usar operadores relacionales y conectores lógicos. Para que una expresión pueda ser usada como una condición, las variables incluidas en la expresión deben tener asignado algún valor, caso contrario será un error pues la condición no podría evaluarse.

Ejemplo. `x<3 and t>2` es una expresión condicional y su valor (**verdadero** o **falso**) dependerá del contenido actual de las variables **x** y **t**

En el lenguaje Python, se pueden usar expresiones con conectores lógicos para verificar en una condición, si una variable pertenece a un intervalo real, como el siguiente ejemplo:

```
x>3 and x<10
```

Estas expresiones también se pueden escribir en forma abreviada:

```
3<x<10
```

Ejemplo. Describa en Python las instrucciones para leer un número real **x**. Verificar si **x** está dentro del intervalo **[2, 5]** y mostrar un mensaje si se cumple esta condición:

```
x=float(input('Ingrese el dato: '))
if 2<=x<=5:
    print('El dato está dentro del intervalo')
```

Ejemplo. Describa en Python un programa para resolver el siguiente problema:

Calcular el valor total que una persona debe pagar por la compra de llantas en un almacén que tiene la siguiente promoción: Si la cantidad de llantas comprada es mayor a **4**, el precio unitario tiene un descuento de **10%.** El programa debe ingresar como datos la catidad de llantas y el precio inicial de cada llanta. Mediante una comparación el programa deberá aplicar el descuento.

Solución

Variables

n: Cantidad de llantas
p: Precio unitario
t: Valor a pagar

Programa

```
#Compra de llantas con descuento
n=int(input('Cantidad de llantas: '))
p=float(input('Precio unitario: '))
if n>4:
     p=0.9*p
t=n*p
print('Valor a pagar: ',t)
```

Prueba del programa

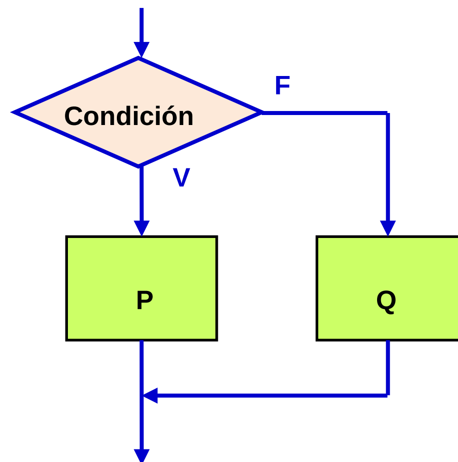
```
>>>
Cantidad de llantas: 3
Precio unitario: 80
Valor a pagar:  240.0

>>>
Cantidad de llantas: 5
Precio unitario: 80
Valor a pagar:  360.0
>>>
```

5.8.2 Ejecución selectiva entre dos bloques de instrucciones

Esta estructura de control de flujo del algoritmo evalúa la condición y dependiendo del resultado realiza las instrucciones incluidas en una de las dos opciones

Representación gráfica



Al entrar a esta estructura, se evalúa la condición. Si el resultado es verdadero (**V**) se ejecutará el bloque **P** asociado al valor verdadero, caso contrario, si el resultado es falso (**F**) se ejecutará el bloque **Q**. El algoritmo continua abajo, despues de ejecutar alguno de los dos bloques.

Seudo lenguaje

```
Si   condición
        P
Sinó
        Q
Fin
```

Lenguaje Python

```
if   condición:
        instrucción en el bloque  P
        instrucción en el bloque  P
                . . .
        instrucción en el bloque  P
else:
        instrucción en el bloque  Q
        instrucción en el bloque  Q
                . . .
        instrucción en el bloque  Q
```

Para definir el bloque de instrucciones asociado a cada una de las dos opciones, se deben escribir las instrucciones desplazadas al menos una columna a la derecha. Todas las instrucciones deben tener el mismo encolumnamiento. Python sugiere el encolumnamiento al momento de escribir las instrucciones del programa. Este encolumnamiento es un deplazamiento de cuatro espacios a la derecha y es el que se usa en este documento.

Ejemplo. Describa en notación algorítmica como asignar a la variable **m** el mayor entre dos valores almacenados respectivamente en las variables **a** y **b**

```
if a>=b:
    m=a
else:
    m=b
```

Ejemplo. Describa en lenguaje Python un programa para resolver el siguiente problema:

Para el pago semanal a un obrero se consideran los siguientes datos: horas trabajadas, tarifa por hora y descuentos. Si la cantidad de horas trabajadas en la semana es mayor a 40, se le debe pagar las horas en exceso con una bonificación de 50% adicional al pago normal.

Solución

Variables

c: Cantidad de horas trabajadas en la semana
t: Tarifa por hora
d: Descuentos que se aplican al pago semanal
p: Pago que recibe el obrero

Programa

```
#Cálculo del pago semanal
c=float(input('Horas trabajadas: '))
t=float(input('Tarifa por hora: '))
d=float(input('Descuentos '))
if c<=40:
     p=c*t - d
else:
     p=40*t + 1.5*t*(c - 40) - d
print('Valor a pagar', p)
```

Prueba del programa

```
>>>
Horas trabajadas: 45
Tarifa por hora: 4
Descuentos 40
Valor a pagar 150.0
```

5.8.3 Decisiones anidadas

En problemas con decisiones múltiples se deben elegir acciones mediante condiciones que deben verificarse en forma sucesiva. Para describir en Python la selección de acciones mediante condiciones se las puede estructurar con decisiones incluidas dentro de otras decisiones con el siguiente formato, el cual establece una jerarquía de acciones:

```
if  condición 1:
    Instrucciones
else:
      if  condición 2:
          Instrucciones
      else:
            if  condiciòn 3:
                Instrucciones
            else:
            . . .
```

La ejecución se realiza de arriba hacia abajo. Si se cumple alguna condición de la cláusula **if**, la ejecución continuá en ese bloque, caso contrario, la ejecución continúa en las instrucciones incluidas en la cláusula **else**. Cada cláusula **if** y cada cláusula **else** pueden abrir otra ruta de decisiones.

Las instrucciones están condicionadas mediante el encolumnamiento asociado a cada cláusula **if** y a cada cláusula **else**.

Es necesario encolumnar las instrucciones con cuidado para establecer con claridad la dependencia a cada cláusula **if** y a cada cláusula **else**.

El algoritmo continua abajo al completarse la ruta de las decisiones.

Ejemplo. Describa en Python un programa para resolver el siguiente problema:

Durante el semestre un estudiante debe realizar tres evaluaciones. Cada evaluación tiene una calificación y la nota total que recibe el estudiante es la suma de las dos mejores calificaciones

Escriba un programa que lea las tres calificaciones y determine cual es la calificación total que recibirá el estudiante.

Solución

Variables

a, b, c: Variables que recibirán los datos de las tres calificaciones
t: Variable con la suma de las dos mejores calificaciones

Solamente hay tres casos posibles y son excluyentes, por lo que se usarán dos decisiones anidadas para verificar dos casos y el tercero será la cláusula else.

Programa

```
# Suma de calificaciones
a=int(input('Ingrese su primera calificación: '))
b=int(input('Ingrese su segunda calificación: '))
c=int(input('Ingrese su tercera calificación: '))
if a>=c and b>=c:
    t=a+b
else:
    if a>=b and c>=b:
        t=a+c
    else:
        t=b+c
print('Su calificación total es: ',t)
```

Prueba del programa

```
>>>
Ingrese su primera calificación: 75
Ingrese su segunda calificación: 68
Ingrese su tercera calificación: 82
Su calificación total es:  157
>>>
```

Ejemplo. Describa en Python la siguiente decisión para pagar una cuenta en un restaurante: Si la cuenta es menor a $50 pago en efectivo. Sinó, si es de $50 hasta $100 pagaré con el celular(dinero electrónico). Pero si es mayor a 100 hasta $200, usaré la tarjeta de débito. Caso contrario, pagaré con la tarjeta de crédito.

Solución

Variables

c: Valor de la cuenta a pagar

En la solución se usarán decisiones anidadas para seleccionar el caso que corresponda al valor de la cuenta. También se usará una forma abreviada que permite Python para expresar elrango para una variable en la condición.

Programa

```
# Pago de una cuenta
c=float(input('Ingrese el valor de la cuenta: '))
if c<50:
    print('Pago en efectivo')
else:
    if 50<=c<=100:
        print('Pago con el celular (dinero electrónico)')
    else:
        if 100<=c<=200:
            print('Pago con la tarjeta de débito')
        else:
            print('Pago con la tarjeta de crédito')
```

Prueba del programa

```
>>>
Ingrese el valor de la cuenta: 120.50
Pago con la tarjeta de débito
>>>
```

5.8.4 Decisiones consecutivas

Esta es otra manera de estructurar decisiones múltiples. Si las decisiones utilizan condiciones similares y con valores diferentes, se las puede estructurar en forma vertical alineadas. Esta estructura es más clara que las decisones anidadas que requieren encolumnar las instrucciones en forma diferente para definir la jerarquía de las decisiones.

El formato de las decisiones consecutivas en Python es:

```
if    condición 1:
      Instrucciónes
elif  condición 2:
      Instrucciones
elif  condición 3:
      Instrucciones
. . .
else:
      Instrucciones
```

Las instrucciones condicionadas están definidas mediante el encolumnamiento asociado a cada condición: condición 1, condición 2, condición 3, etc., Si se cumple alguna de esta condiciones, se ejecutan las instrucciones condicionadas. Si no se cumple alguna de estas condiciones, se ejecutaràn las instrucciones asociadas a la cláusula **else**. Este último componente es opcional.

La ejecución continua abajo de esta estuctura, despues de ejecutar alguno de los bloques de instrucciones.

En general, los problemas con decisiones múltiples pueden resolverse con cualquiera de estos dos tipos decisiones: decisiones anidadas o decisiones consecutivas.

Ejemplo. Describa un programa para resolver el siguiente problema:

Leer el número de llantas de una compra y mostrar el valor que debe pagarse. El almacén las vende con la siguiente política: Si se compran menos de 5 llantas, el precio unitario es 80. Si se compran 6 o 7, el precio unitario es 70, y si se compran más de 7 llantas, el precio unitario es 60.

Solución

Variables

- **n**: Cantidad de llantas compradas
- **p**: Precio unitario (80, 70, o 60)
- **t**: Valor de la compra

Programa

```
# Compra de llantas con descuento
n=int(input('Cantidad de llantas: '))
if n<5:
    p=80
elif n==5 or n==6:
    p=70
else:
    p=60
t=n*p
print('Valor a pagar: ', t)
```

Prueba del programa

```
>>>
Cantidad de llantas: 6
Valor a pagar:  420
>>>
Cantidad de llantas: 4
Valor a pagar:  320
>>>
Cantidad de llantas: 8
Valor a pagar:  480
>>>
```

Ejemplo. El precio de una pizza depende de su tamaño según la siguiente tabla:

Tamaño	Precio
1	$5
2	$8
3	$12

Cada ingrediente adicional cuesta $1.5.

Escriba un programa en Python que lea el tamaño de la pizza y el número de ingredientes adicionales y muestre el precio que debe pagar

Solución

Variables

t: Tamaño de pizza
n: Número de ingredientes
p: Valor a pagar

Programa

```
#Compra de una pizza
t=int(input('Tamaño de la pizza: '))
n=int(input('Número de ingredientes adicionales: '))
if t==1:
      p=5+1.5*n
elif t==2:
      p=8+1.5*n
elif t==3:
      p=12+1.5*n
else:
      p=0
print('Valor a pagar: ', p)
```

Prueba del programa

```
>>>
Tamaño de la pizza: 2
Número de ingredientes adicionales: 3
Valor a pagar:  12.5
>>>
```

Ejemplo. Use decisiones consecutivas para resolver el ejemplo del pago de la cuenta en un restaurante revisado en la sección anterior.

Variables

c: Valor de la cuenta a pagar

En la solución se usarán decisiones consecutivas para seleccionar el caso respectivo

Programa

```
# Pago de una cuenta
c=float(input('Ingrese el valor de la cuenta: '))
if c<50:
    print('Pago en efectivo')
elif 50<=c<=100:
    print('Pago con el celular (dinero electrónico)')
elif 100<=c<=200:
    print('Pago con la tarjeta de débito')
else:
    print('Pago con la tarjeta de crédito')
```

Prueba del programa

```
>>>
Ingrese el valor de la cuenta: 120.50
Pago con la tarjeta de débito
>>>
```

5.8.5 Ejercicios de programación con decisiones

Para cada ejercicio escriba y pruebe un programa en la ventana de edición de Python.

1. Dados el radio y altura de un cilindro, si la altura es mayor al radio calcule y muestre el valor del volumen del cilindro, caso contrario muestre el valor del área del cilindro.

2. Dados el radio y altura de un cilindro, si la altura es mayor al radio calcule y muestre el valor del volumen, caso contrario muestre el mensaje: 'Error'

3. Dadas las dimensiones de un bloque rectangular, calcule las diagonales de las tres caras diferentes. Muestre el valor de la mayor diagonal.

4. Lea un número de dos cifras. Determinar si la suma de ambas cifras es un número par o impar. Muestre un mensaje

5. Lea un número. Determine si es entero y múltiplo de 7

6. Lea la cantidad de Kw que ha consumido una familia y el precio por Kw. Si la cantidad es mayor a 700, incremente el precio en 5% para el exceso de Kw sobre 700. Muestre el valor total a pagar.

7. Lea un valor de temperatura **t** y un código **p** que puede ser **1** o **2**. Si el código es **1** convierta la temperatura **t** de grados **f** a grados **c** con la fórmula **c=5/9(t-32).** Si el código es **2** convierta la temperatura **t** de grados **c** a **f** con la fórmula: **f=32+9t/5**, caso contrario muestre un mensaje de error.

8. Dadas las tres calificaciones de dos estudiantes. La calificación final de cada uno es la suma de sus dos mejores calificaciones. Muestre un mensaje que indique cual estudiante (1 o 2) tiene la mayor calificación final.

9. Dadas las tres calificaciones de un estudiante, encuentre y muestre la calificación myor y la califiicación menor

10. Dados los tres lados de un triángulo determine su tipo: Equilátero, Isòsceles, o Escaleno

11. Dadas la abscisa y ordenada de dos puntos, calcule su distancia al origen y determine cual de los dos puntos (primero o segundo) está más cerca del origen. La respuesta deberá ser un mensaje: ‘Punto 1’ o ‘Punto 2’

Punto	Abscisa	Ordenada
1	a	b
2	c	d

Fórmula de la distancia del punto (**x, y**) al origen: $\sqrt{x^2 + y^2}$

12. Lea la cantidad de Kw que ha consumido una familia y el precio por Kw. Si la cantidad es mayor a 700, incremente el precio en 5% para el exceso de Kw sobre 700. Muestre el valor total a pagar.

13. Juan, Pedro y José trabajan en una empresa que paga semanalmente. Ingrese para cada uno los siguientes datos del trabajo semanal: horas trabajadas, salario por hora, y descuentos. Calcule el pago semanal que recibirá cada uno y determine cual de los tres recibirá el mayor pago semanal. No considere el pago de horas extras.

14. Lea las dimensiones de un bloque rectangular (largo, ancho y altura del bloque), y el diámetro de un agujero. Determine si es posible que el bloque pueda pasar por el agujero.

Sugerencia: Calcule cada una de las tres diagonales del bloque. Si alguna de ellas tiene un valor menor al diámetro del agujero muestre el mensaje: **‘Si pasa por el agujero’.**

15. Un código de tres cifras debe cumplir la siguiente regla para que sea vàlido: La tercera cifra debe ser igual al mòdulo 10 del producto de las dos primeras cifras. Escriba un programa que lea un código y verifique si cumple la regla anterior. Muestre un mensaje correspondiente.

Ejemplo. 384 es un código vàlido pues el módulo de 3x8 en 10 es igual a 4

16. El número de pulsaciones que debe tener una persona por cada 10 segundos de ejercicio aeróbico se calcula con la fórmula:

Género femenino (1): número de pulsaciones = (220 - edad en años)/10
Género masculino (2): número de pulsaciones = (210 - edad en años)/10

Lea la edad y el género y muestre el número de pulsaciones.

17. El índice de masa corporal IMC de una persona se calcula con la fórmula $IMC=P/T^2$ en donde P es el peso en Kg. y T es la talla en metros.

Lea un valor de P y de T, calcule el IMC y muestre su estado según la siguiente tabla:

IMC	**Estado**
Menos de 18.5	Desnutrido
[18.5 a 25.5]	Peso Normal
Más de 25.5	Sobrepeso

18. Otro reporte de salud muestra una tabla diferente del índice de masa corporal IMC de una persona que se calcula con la fórmula $IMC=P/T^2$ en donde P es el peso en Kg. y T es la talla en metros.

Lea un valor de P y de T, calcule el IMC y muestre su estado según la siguiente tabla:

IMC	**Estado**
Menor a 20	Desnutrido
[20, 25)	Normal
[25,30)	Sobrepeso
[30,35)	Obesidad Grado 1
[35,40)	Obesidad Grado 2
Mayor a 40	Obesidad Grado 3

19. En un concurso hay tres jueces. La opinión del juez es 1 si está a favor y 0 si está en contra. Para que un participante pueda continuar en el concurso debe tener al menos dos votos favorables. Escriba un algoritmo que lea los votos de los tres jueces y muestre el resultado mediante un mensaje: CONTINUA o ELIMINADO. No sume votos. Debe compararlos.

20. Dadas las dimensiones de un bloque rectangular, calcule y muestre el área de la cara de mayor dimensión.

21. Se conocen tres de los cuatro números de una matriz cuadrada de tamaño 2. Lea estos tres números y determine cual debe ser el cuarto número para que el determinante de la matriz sea igual a 0.

22. Lea un número **x** y los números **a, b**. Suponga que **a<b**. y que **x ≠ a**, **x ≠ b**. Determine en que lugar se encuentra el número **x**, antes de **a**, entre **a** y **b** o después de **b**. Muestre un mensaje.

23. Lea las tres calificaciones que obtuvo un estudiante en una materia. No suponga que estos tres números están ordenados. Describa como ordenarlos en forma ascendente y muestre los números ordenados

24. Lea los números de matrícula de tres estudiantes que toman la materia A y los números de matrícula de tres estudiantes que toman la materia B. Encuentre cuantos estudiantes que toman la materia A, también toman la materia B.

25. Un almacén ofrece un producto con descuento según la siguiente tabla:

Kilos comprados	**Precio por kilo**
Menos de 3	$2.4
[3 a 6)	$2.3
[6 a 10)	$2.1
Más de 10	$1.85

Lea la cantidad comprada y determine cuanto debe pagar.

26. Modifique el siguiente algoritmo de tal manera que las condiciones que contienen los conectores lógicos (∧, ∨, ⌉ **)** sean sustituidas con instrucciones en las cuales las condiciones no tengan estos conectores lógicos. Ambos algoritmos debe ser equivalentes.

```
Leer a, b, c
x ← 0
y ← 0
Si ⌉(a<2 ∧ b>1) ∨ (c>3)
   x ← a + b
Sino
   y ← b - c
Fin
Mostrar x, y
```

27. Considere el siguiente algoritmo

```
1  Leer  a, b, c
2  Si a<b salte a la línea 4
3  Salte a la línea 5
4  Si b>c salte a la línea 6
5  Salte a la línea 7
6  Mostrar a
7  Mostrar b
8  Mostrar c
```

a) Construya un diagrama de flujo ordenado y simplificado que sea equivalente al algoritmo propuesto.

b) Interprete el diagrama de flujo y codifíquelo en notación Python

5.9 Números aleatorios

Los números aleatorios son valores que están en un rango de posibles resultados pero no se puede predecir cual es el resultado que se obtendrá al tomar alguno. Por ejemplo, al lanzar un dado no podemos afirmar cual resultado se obtendrá entre los seis números posibles.

Los lenguajes computacionales tienen funciones especiales para generar números aleatorios, los cuales son útiles para muchas aplicaciones de interés.

El módulo **random** es una librería de Python que contiene funciones para generar números aleatorios. Para acceder a este módulo especial se lo debe cargar con la siguiente directiva:

```
from random import*
```

Algunas funciones básicas del módulo **random:**

a) Generar un número aleatorio real en el intervalo semi abierto **[0, 1)**

```
random()
```

b) Generar un número aleatorio entero en el intervalo **[a, b]** incluyendo los extremos

```
randint(a,b)
```

c) Iniciar una secuencia de números aleatorios con una semilla dada

```
seed(n)
```

La semilla **n**, es un entero positivo. Es un valor que usa el generador de números aleatorios para construir la secuencia. El uso de **seed** es opcional. Si se especifica debe escribirse al inicio. Esto hará que la secuencia se repita en cada prueba.

d) Elegir aleatoriamente un elemento de una lista

```
choice(s)
```

e) Desordenar aleatoriamente los elementos de una lista

```
shuffle(s)
```

En estas instrucciones **s** representa una lista cuyos elementos pueden tener diferente tipo, como se verá en una sección más adelante.

f) Generar una número aleatorio con distribución Normal o Gaussiana

```
gauss(mu,sigma)
```

mu es la media, **sigma** es la desviación estándar

g) Obtener una muestra de **k** elementos diferentes de una lista (población) **p**
Las listas se estudiarán en un capitulo posterior.

```
sample(p,k)
```

Ejemplos. Uso de las funciones del módulo **random** en la ventana interactiva

```
>>> from random import*
>>> x=random()
>>> x
0.9635612618951365
>>> r=randint(1,6)
>>> r
3
>>> s=[10,90,20,40,50]                Crear una lista
>>> shuffle(s)                        Desordenar la lista
>>> s
[40, 50, 10, 20, 90]
>>> c=choice(s)
>>> c
50
>>> v=gauss(10,2)
>>> v
9.31986340769587
>>>
```

Ejemplo. Escriba un programa que genere un número aleatorio de un dado. Si sale 6 muestre el mensaje: 'Afortunado', caso contrario muestre el número que se obtuvo y el mensaje: 'No hubo suerte hoy'

Solución

Variable

n: número aleatorio entre 1 y 6

Programa

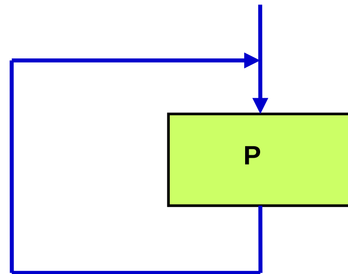
```
# Número de un dado
from random import *
n=randint(1, 6)
if n==6:
    print('Afortunado')
else:
    print('Salió: ',n)
    print('No hubo suerte hoy')
```

Prueba del programa

```
>>>
Salió 3
No hubo suerte hoy
>>>
```

5.10 Ciclos en Python

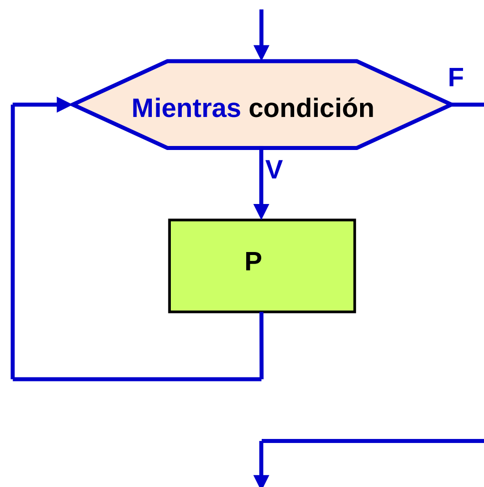
Estas estructuras de control se usan para describir la ejecución repetida de un bloque de instrucciones. El objetivo es colocar el bloque de instruciones dentro de un ciclo como se muestra en el gráfico. Sin embargo, es necesario agregar un dispositivo que permita salir del ciclo para que el algoritmo pueda continuar:



Hay tres formas comunes que se usan para salir de una estructura de repetición. Dos de ellas usan una condición para salir del ciclo. Esta condición puede estar al inicio o al final. La otra forma, utiliza los valores de un conteo o una secuencia de valores para controlar la repetición.

5.10.1 Ejecución repetida de un bloque mediante una condición al inicio

Representación gráfica



Al entrar a esta estructura, se evalúa la condición. Si el resultado es verdadero (**V**) se ejecutarán las instrucciones en el bloque y regresará nuevamente a evaluar la condición.

Mientras la condición mantenga el valor verdadero el bloque de instrucciones seguirá ejecutándose. Esto significa que es necesario que en algún ciclo la condición tome el resultado falso (**F**) para salir de la estructura y continuar la ejecución después del bloque.

La condición es cualquier expresión cuyo resultado puede ser únicamente verdadero (**V**) o falso (**F**). Puede incluir operadores para comparar el contenido de variables y también se pueden usar los conectores de la lógica matemática.

Seudo lenguaje

```
Mientras  condición
      P
Fin
```

Lenguaje Python

```
while  condición:
       Instrucción en el bloque  P
       Instrucción en el bloque  P
       . . .
       Instrucción en el bloque  P
```

El bloque de las instrucciones que se repetirán está definido mediante el encolumnamiento. Las instrucciones que se repetirán deben escribirse desplazadas algunas columnas a la derecha de la palabra **while.** Todas deben tener el mismo encolumnamiento**.**

Ejemplo. Simular lanzamientos de un dado. Muestre el resultado en cada intento. Finalice cuando salga el número **3**.

Solución

Variable

x: resultado del dado en cada lanzamiento. Inicialmente un valor nulo para entrar al ciclo. Note este artificio útil para usar la estructura **while**

Programa

```
#Simular el lanzamiento de un dado hasta que salga el número 3
from random import *
x=0
while x!=3:
    x=randint(1, 6)
    print(x)
```

Prueba del programa

```
>>>
2
6
6
1
6
4
3
```

El uso de la estructura **while** en este ejemplo es natural pues no se conoce la cantidad de repeticiones que deben realizarse para que se cumpla la condición y pueda salir del ciclo.

Nota. Observe el siguiente programa que incluye un cambio menor respecto al programa anterior. ¿Cual es el resultado que se obtendrá?

```
#Simular el lanzamiento de un dado hasta que sale un número
from random import *
x=0
while x!=3:
    x=randint(1, 6)
print(x)
```

Respuesta: El encolumnamiento de la instrucción **print** la saca del bloque que se repite en el ciclo. Por lo tanto solamente se mostrará el valor final de la variable **x**.

Si es de interés conocer el número de repeticiones realizadas se debe incluir una variable para efectuar el conteo como se muestra en el siguiente ejemplo.

Ejemplo. Simular lanzamientos de un dado. Determinar la cantidad de lanzamientos que se realizaron hasta que se obtuvo el número **3**.

Solución

Variables

- **x:** Resultado del dado en cada lanzamiento. Al inicio el valor 0 para entrar al ciclo
- **c:** Conteo de repeticiones

Programa

```
#Conteo de lanzamientos de un dado
from random import *
c=0
x=0
while x!=3:
    x=randint(1, 6)
    c=c+1
print('Cantidad de lanzamientos hasta que salió el 3: ', c)
```

Prueba del programa

```
>>>
Cantidad de lanzamientos hasta que salió el 3:  5
>>>
```

Ejemplo. Suma de los cuadrados de los primeros números naturales

Solución

Variables

n: dato (número natural hasta el que se llegará)
s: suma de cuadrados
i: cada número natural

Programa

```
#Suma de cuadrados
n=int(input('Ingrese el valor final: '))
s=0
i=1
while i<=n:
    c=i**2
    s=s+c
    i=i+1
print('La suma es: ', s)
```

Prueba del programa

```
>>>
Ingrese el valor final: 20
La suma es:  2870
```

Ejemplo. Describa en notación Python una solución para el siguiente problema usando la estructura de ciclos condicionada.

En un cultivo se tiene una cantidad inicial de bacterias. Cada día esta cantidad se duplica. Determine en que día la cantidad excede a un valor máximo.

Solución

Variables

x: Cantidad inicial de bacterias
m: Cantidad máxima de bacterias
d: Día

Programa

```
#Crecimiento de la cantidad de bacterias
x=int(input('Ingrese la cantidad inicial  '))
m=int(input('Ingrese la cantidad máxima '))
d=0;
while x<=m:
    x=2*x
    d=d+1
print('La cantidad excede al máximo en el dia:  ', d)
```

Prueba del programa

```
>>>
Ingrese la cantidad inicial  200
Ingrese la cantidad máxima  5000
La cantidad excede al máximo en el dia:   5
```

Ejemplo. Dado un número entero, genere una secuencia numérica con la siguiente regla. Esta secuencia se denomina de Ulam. Esta secuencia siempre llega al número 1

$$x = \begin{cases} x/2, & x \text{ par} \\ 3x+1, & x \text{ impar} \end{cases}$$

Una prueba manual de esta definición

x	
20	Valor inicial
10	par
5	impar
16	par
8	par
4	par
2	par
1	valor final

Solución

Variable

x: número entero positivo

Programa

```
# Secuencia de Ulam
x=int(input('Ingrese el dato inicial: '))
while x>1:
    if x%2 == 0:
        x=x//2
    else:
        x=3*x+1
    print(x)
```

Este ejemplo muestra un tema de interés: una estructura de decisión dentro de un ciclo.

Observe el encolumnamiento de las instrucciones que define cuales pertenecen a cada estructura de control. El encolumnamiento de instrucciones es obligatorio en el lenguaje Python.

Prueba del programa

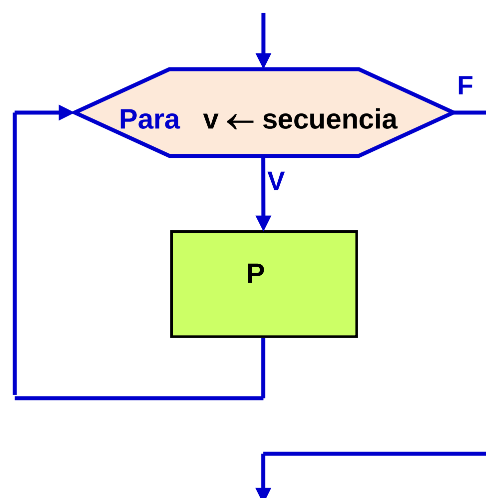
```
>>>
Ingrese el dato inicial: 20
10
5
16
8
4
2
1
>>>
```

5.10.2 Ejecución repetida de un bloque mediante una secuencia de valores

Esta es forma es muy usada para controlar la repetición de bloques de instrucciones.

Esta estructura controla el ciclo mediante una secuencia de valores.

Representación gráfica



Para usar esta estructura de repetición es necesario especificar una variable y una lista o secuencia de los valores que puede tomar. El ciclo se repetirá con cada valor especificado para la variable.

Al entrar a esta estructura se inicia la variable de control. Con cada valor que toma esta variable se ejecuta el bloque de instrucciones. A continuación, el flujo regresa nuevamente al inicio del ciclo y la variable toma el siguiente valor de la secuencia. Cuando el valor de la variable llegue al valor final, el ciclo finalizará y la ejecución continuará después del bloque.

La variable de control del ciclo puede especificarse con alguna notación que exprese cuales son los valores que puede tomar.

Seudo lenguaje

Para **v ← secuencia**

P

Fin

Lenguaje Python

Existen dos formas de especificar la repetición controiada por una secuencia en el lenguaje de Python:

a) Secuencia definida mediante una lista

Se puede especificar una **secuencia** mediante una lista de valores escritos entre corchetes **[ ]** o entre paréntesis **( )** con el siguiente formato:

```
for  v  in secuencia:
      instrucción en el bloque P
      instrucción en el bloque P
      . . .
      instrucción en el bloque P
```

En donde **v** es la variable que recorre la secuencia y **P** es el bloque que contiene las instrucciones que se desea repetir.

El bloque de las instrucciones que se repetirán está definido mediante el encolumnamiento. Las instrucciones que se repetirán deben escribirse desplazadas algunas columnas a la derecha de la palabra **for**. Todas deben tener el mismo encolumnamiento**.**

Ejemplos. Las siguientes listas son ejemplos de secuencias de control para el ciclo for:

```
[2, 5, 7, 8, 9, 5, 4]

['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo']

[3.5, 2.8, 0.75, 4.3, 7.6]
```

Ejemplo. Describa un ciclo para mostrar cada entero entre 1 y 5 junto con su cuadrado

```
for n in [1, 2, 3, 4, 5]:
    c=n**2
    print(n,c)
```

Prueba del programa

```
>>>
1 1
2 4
3 9
4 16
5 25
```

Los elementos de la lista pueden estar en cualquier orden y pueden estar repetidos:

```
for n in [3, 2, 5, 1, 5, 4]:
    c=n**2
    print(n,c)
```

Prueba del programa

```
>>>
3 9
2 4
5 25
1 1
5 25
4 16
```

Ejemplo. Describa un ciclo que muestre cada nombre de las provincias de la costa ecuatoriana.

```
for p in ['Guayas','Los Rios','El Oro', 'Manabí',
          'Sta. Elena','Esmeraldas']:
    print('Provincia: ',p)
```

Prueba del programa

```
>>>
Provincia:  Guayas
Provincia:  Los Rios
Provincia:  El Oro
Provincia:  Manabí
Provincia:  Sta. Elena
Provincia:  Esmeraldas
```

b) Secuencia definida mediante un rango

La forma más utilizada para especificar secuencias de control para la estructura **for** se define con la función **range** con la cual se especifica un rango para los valores con el siguiente formato:

```
for  v  in range(especificación):
      instrucción en el bloque P
      instrucción en el bloque P
      . . .
      instrucción en el bloque P
```

En donde **v** es la variable que tomará cada valor del rango especificado, y **P** es el bloque que contiene las instrucciones que se desean repetir.
El rango se debe especificar únicamente con **números enteros,** no se pueden usar decimales o caracteres

La especificación del rango puede tomar varias formas como se muestra en los ejemplos:

El rango se puede especificar con el **valor final**. En este caso, el valor inicial es cero.

Ejemplo.

`range(10)`	Genera los valores **0, 1, 2, 3, 4, 5, 6, 7, 8, 9** El ciclo se repetirá **10** veces.

El rango se puede especificar con los **extremos**

Ejemplos.

`range(2,10)`	Genera los valores **2, 3, 4, 5, 6, 7, 8, 9** El ciclo se repetirá **8** veces
`range(4,1)`	El valor inicial excede al valor final del rango. No se pueden generar valores. El ciclo no se realiza ni una vez.

> La secuencia generada con **range** no incluye el extremo final del rango.

El rango se puede especificar con los **extremos** y el **incremento**

Ejemplos.

`range(1,15,2)`	Genera los valores **1, 3, 5, 7, 9, 11, 13** El ciclo se repetirá **7** veces.
`range(8,1,-1)`	Genera los valores **8, 7, 6, 5, 4, 3, 2** El ciclo se repetirá **7** veces.

Ejemplo. Describa un ciclo para mostrar cada entero entre 1 y 5 junto con su cuadrado:

```
for n in range(1,6):
    c=n**2
    print('Número: ',n,' Cuadrado: ',c)
```

Prueba del programa

```
>>>
Número:  1  Cuadrado:  1
Número:  2  Cuadrado:  4
Número:  3  Cuadrado:  9
Número:  4  Cuadrado:  16
Número:  5  Cuadrado:  25
```

Ejemplo. Se necesita sumar los cuadrados de los primeros **n** números naturales.

Solución

Variables

n: número final
i: cada número natural
s: suma de los cuadrados

Programa

```
#Suma de cuadrados
n=int(input('Ingrese el valor final: '))
s=0
for i in range(1,n+1):
    c=i**2
    s=s+c
print('La suma es: ',s)
```

Prueba del programa

```
>>>
Ingrese el valor final: 100
La suma es:  338350
>>>
```

Ejemplo. Calcule y muestre el promedio de un grupo de datos ingresados desde el teclado

Solución

Variables

n: cantidad de datos
i: conteo de ciclos
x: cada dato ingresado desde el teclado
s: suma de los datos
p: promedio

Programa

```
#Promedio de un grupo de datos
n=int(input('Cantidad de datos:  '))
s=0
for i in range(n):
    x=float(input('Ingrese el siguiente dato:  '))
    s=s+x
p=s/n
print('El promedio es: ',p)
```

Prueba del programa

```
>>>
Cantidad de datos: 5
Ingrese el siguiente dato: 4.2
Ingrese el siguiente dato: 5.8
Ingrese el siguiente dato: 2.5
Ingrese el siguiente dato: 8.1
Ingrese el siguiente dato: 6.2
El promedio es:  5.36
>>>
```

Ejemplo. Describa en notación Python una solución al siguiente problema: Dado un entero positivo **n**, se desea verificar que la suma de los primeros **n** números impares es igual a n^2

Ej. n = 5: 1 + 3 + 5 + 7 + 9 = 5^2

Solución

Variables

n: Cantidad de números impares
k: Cada número impar
s: Suma de impares
i: Conteo de ciclos

```
# Suma de impares
n=int(input('Ingrese la cantidad de impares:  '))
s=0
for i in range(1,n+1):
    k=2*i-1
    s=s+k
if s==n**2:
    print('Verdadero')
else:
    print('Falso')
```

Prueba del programa

```
>>>
Ingrese la cantidad de impares:  5
Verdadero
>>>
```

Se especifica el final del rango con el valor **n+1** para que se incluya en el ciclo el valor **n**

Note que este resultado solo prueba que la propiedad se cumple con el dato ingresado .
No es una demostración de que la propiedad es verdadera con todos los enteros positivos.

Ejemplo. Lea un grupo de datos (precios) desde el teclado. Encuentre y muestre el mayor valor

Solución

Variables

n: cantidad de datos
i: conteo de ciclos
x: cada dato ingresado desde el teclado
t: el mayor valor

La variable **t** que contendrá el mayor valor se la inicia con cero. En un ciclo se ingresará cada dato y se lo comparará con **t,** si es mayor**,** la variable **t** es asignada con el valor del dato ingresado. Al finalizar el ciclo **t** contendra el mayor valor

Programa

```
#El mayor valor de un grupo de datos
n=int(input('Cantidad de datos: '))
t=0
for i in range(n):
    x=float(input('Ingrese el siguiente dato: '))
    if x>t:
        t=x
print('El mayor valor es: ',t)
```

Prueba del programa

```
>>>
Cantidad de datos: 5
Ingrese el siguiente dato: 45
Ingrese el siguiente dato: 37
Ingrese el siguiente dato: 28
Ingrese el siguiente dato: 56
Ingrese el siguiente dato: 24
El mayor valor es:  56.0
>>>
```

Ejemplo. Dado un número entero positivo, encuentre todos sus divisores enteros positivos.

Solución

Variables

n: número entero positivo (dato)
d: cada número entero entre 1 y **n**, posible divisor de **n**

Programa

```
#Divisores de un entero
n=int(input('Ingrese un entero positivo: '))
for d in range (1,n+1):
    if n%d==0:
        print('Divisor: ',d)
```

Prueba del programa

```
>>>
Ingrese un entero positivo: 20
Divisor:  1
Divisor:  2
Divisor:  4
Divisor:  5
Divisor:  10
Divisor:  20
>>>
```

Se especifica el final del rango con el valor **n+1** para que en el ciclo se incluya el valor **n**

Ejemplo. Dado un número entero, cuente sus divisores enteros exactos.

Solución

Variables

n: número entero positivo (dato)
x: cada número entero entre 1 y n, posible divisor de n
c: cantidad de divisores

Programa

```
#Divisores de un entero
n=int(input('Ingrese un entero positivo: '))
c=0
for d in range (1,n+1):
    if n%d==0:
        c=c+1
print('Cantidad de divisores: ',c)
```

Prueba del programa

```
>>>
Ingrese un entero positivo: 20
Cantidad de divisores:  6
>>>
```

Ejemplo. Dado un número entero determine si es un número **primo**

Solución

Variables

n: número entero positivo (dato)
x: cada número entero entre 1 y n, posible divisor de n
c: cantidad de divisores

Si el conteo de divisores enteros es mayor a 2, el número no es primo

Programa

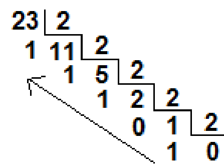
```
#Determinar si un número es primo
n=int(input('Ingrese un entero positivo: '))
c=0
for d in range (1,n+1):
    if n%d==0:
        c=c+1
if c>2:
    print('No es primo')
else:
    print('Si es primo')
```

Prueba del programa

```
>>>
Ingrese un entero positivo: 43
Si es primo
>>>
Ingrese un entero positivo: 57
No es primo
>>>
```

Ejemplo. Dado un número entero encuentre los dígitos de su equivalente en el sistema binario. El algoritmo para obtener los dígitos binarios de un número entero decimal consiste en dividirlo sucesivamente para 2. Los resíduos de la división entera tomados desde el final hacia arriba son los dígitos buscados.

Para probar: Obtener los dígitos binarios del número **23**



Entonces, **23** es equivalente a **1 0 1 1 1** en el sistema binario

Solución

Variables

n: Número entero positivo
b: Cadena de caracteres que contiene los dígitos de **n** en el sistema binario.
La cadena crece en forma recurrente de derecha a izquierda

```
#Convertir un entero positivo a binario
n=int(input('Ingrese un entero positivo: '))
b=''
while n>0:
    d=n%2
    n=n//2
    b=str(d)+b
print('Número binario: ',b)
```

Prueba del programa

```
Ingrese un entero positivo: 23
Número binario:  10111
>>> b
'10111'
```

La variable b contiene una cadena de caracteres con los dígitos en binario

Para comparar, se usa la función de conversion de tipo de Python

```
>>> n=23
>>> b=bin(n)
>>> b
'0b10111'                    Cadena de caracteres con los dígitos en binario

>>> b=0b10111                Número en binario
>>> n=int(0b10111)
>>> n
23
```

Ejemplo. Simule **n** lanzamientos de un dado. Muestre cuantas veces se obtuvo el **3**.

Solución

Variables

n: dato (cantidad de lanzamientos)
i: conteo de lanzamientos
x: cada número aleatorio
c: cantidad de veces que se obtuvo el 3

Programa

```
#Simular lanzamientos de un dado
from random import*
n=int(input('Cantidad de lanzamientos: '))
c=0
for i in range(n):
    x=randint(1,6)
    if x==3:
        c=c+1
print('Conteo de resultados favorables: ',c)
```

Prueba del programa

```
>>>
Cantidad de lanzamientos: 600
Conteo de resultados favorables:  105
>>>
```

NOTA: Pruebe con valores grandes de **n**. Verifique que la respuesta es aproximadamente **1/6** de **n.** Este resultado confirma lo que indica la Teoría de la Probabilidad. Si el dado es balanceado, cada número tiene aproximadamente la misma probabilidad de salir.

Ejemplo. Simule **n** lanzamientos de dos dados. Muestre cuantas veces los dos dados tuvieron el mismo resultado.

Solución

Variables

n: dato (cantidad de lanzamientos)
i: conteo de lanzamientos
a,b: contendrán números aleatorios enteros entre 1 y 6
c: cantidad de veces en las que **a** fue igual a **b**

Programa

```
#Lanzamientos de dos dados
from random import*
n=int(input('Cantidad de lanzamientos: '))
c=0
for i in range(n):
    a=randint(1,6)
    b=randint(1,6)
    if a==b:
        c=c+1
print('Cantidad de repetidos: ',c)
```

Prueba del programa

```
>>>
Cantidad de lanzamientos:  400
Cantidad de repetidos:  67
>>>
```

Ejemplo. Simular **n** intentos de un juego con un dado, con las siguientes reglas:

Si sale

6	Gana 4 dólares
3	Gana 1 dólar
1	Siga jugando
2,**4** o **5**	Pierde 2 dólares

Solución

Variables

n: dato (cantidad de intentos)
i: conteo de lanzamientos
x: cada número aleatorio
s: cantidad acumulada de dinero

Programa

```
#Simulación de un juego de azar
from random import*
n=int(input('Cantidad de intentos: '))
s=0
for i in range(n):
    x=randint(1,6)
    if x==6:
        s=s+4
    elif x==3:
        s=s+1
    elif x==2 or x==4 or x==5:
        s=s-2
print('Ganancia total: ',s)
```

Prueba del programa

```
>>>
Cantidad de intentos: 100
Ganancia total:  -20
>>>
```

Observe en las pruebas que si el número de intentos es grande, normalmente se obtendrá un valor negativo (pérdida), pues las reglas dadas en el ejemplo están desbalanceadas a favor de perder. La teoría de la probabilidad demuestra estos resultados formalmente con la definición del Valor Esperado. Esto significa que no es una buena idea jugar este juego ni ningún otro juego de azar, pues siempre están desbalanceados a favor del promotor.

Ejemplo. Simule **n** lanzamientos de un dardo en un cuadrado de 1 m de lado. Determine cuantos intentos cayeron dentro de un círculo inscrito en el cuadrado.

Solución

Variables

n: dato (cantidad de intentos)
i: conteo de lanzamientos
x,y: coordenadas para cada lanzamiento (números aleatorios)
c: cantidad de veces que están dentro del círculo de radio 1

Programa

```
#Puntos aleatorios dentro de un círculo
from random import*
n=int(input('Cantidad de intentos: '))
c=0
for i in range(n):
    x=random()
    y=random()
    if x**2 + y**2 <= 1:
        c=c+1
print('Dentro del círculo: ',c)
```

Prueba del programa

```
>>>
Cantidad de intentos: 100
Dentro del círculo:  80
>>>
```

Ejemplo. Dado un conjunto de enteros numerados **1** a **n**, elegir una muestra aleatoria de **m** números, **m≤n**

Solución

Variables

n: Tamaño de la población (dato)
m: Tamaño de la muestra (dato)
i: Variable para el conteo de repeticiones
x: Contiene cada número aleatorio seleccionado para la muestra

Programa

```
#Muestra aleatoria (con repeticiones)
from random import*
n = int(input('Ingrese tamaño de la población '))
m = int(input('Ingrese tamaño de la muestra '))
for i in range(m):
    x=randint(1,n)
    print(x)
```

Prueba del programa

```
>>>
Ingrese tamaño de la población 10
Ingrese tamaño de la muestra 4
3
9
2
3
>>>
```

Note que la muestra puede tener valores repetidos. Posteriormente se describirá una técnica para que una muestra no incluya elementos repetidos.

Ejemplo. Escriba un programa para simular el siguiente juego: una rana es colocada aleatoriamente en la casilla central de una caja cuadriculada de tamaño **9x9** dm. La rana realiza saltos aleatoriamente de **1** dm. en cualquiera de las cuatro direcciones: arriba, abajo, izquierda o derecha. Determine la cantidad de saltos realizados hasta llegar a alguno de los bordes de la caja.

Solución

La casilla central coincidirá con las coordenadas (0, 0)

Variables

x,y: coordenadas de las casillas
d: dirección del salto (aleatoria)
i: conteo de intentos

Programa

```
from random import*
x=0
y=0
i=0
while -5<x<5 and -5<y<5:
    d=randint(1,4)
    i=i+1
    if d==1:
        x=x+1
    elif d==2:
        x=x-1
    elif d==3:
        y=y+1
    else:
        y=y-1
    print(x,y)
print('Cantidad de intentos: ',i)
```

Prueba del programa

```
>>>
 1 0
 1 1
 0 1
-1 1
-2 1
-2 0
-2 1
-2 2
-2 3
-3 3
-3 2
-4 2
-4 1
-5 1
Cantidad de intentos:  14
```

Ejemplo. Escriba un programa para representar mediante barras de asteriscos, 10 números aleatorios con valores enteros entre 1 y 20.

Solución

Variables

i: variable para el conteo de repeticiones
n: número aleatorio
barra: barra de n asteriscos

Programa

```
from random import*
for i in range(10):
    n=randint(1,20)
    barra=''
    for j in range(1,n+1):
        barra=barra+'*'
    print('%4d '%n,barra)
```

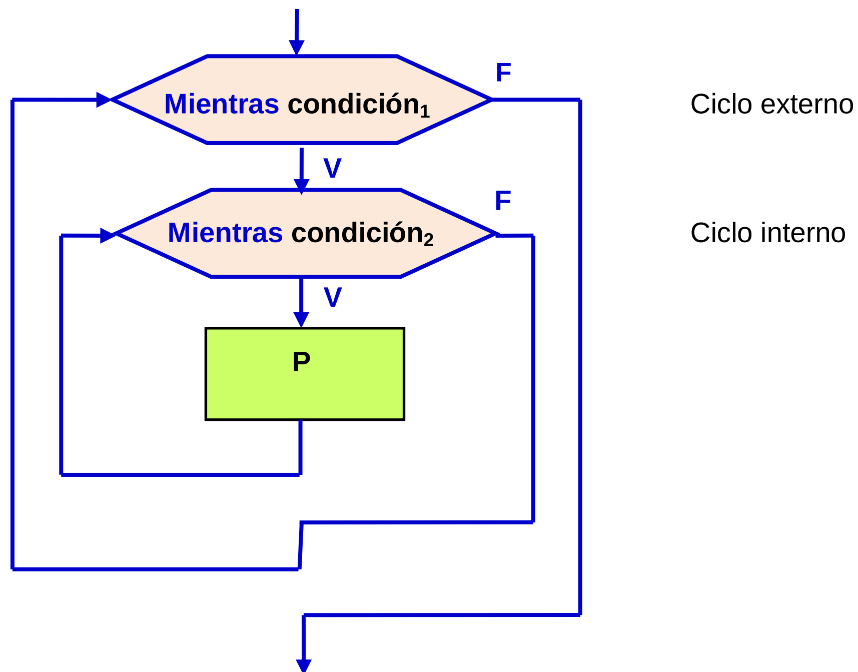
Prueba del programa

```
>>>
   3  ***
  11  ***********
  13  *************
  12  ************
  10  **********
   4  ****
   4  ****
   6  ******
   5  *****
  17  *****************
```

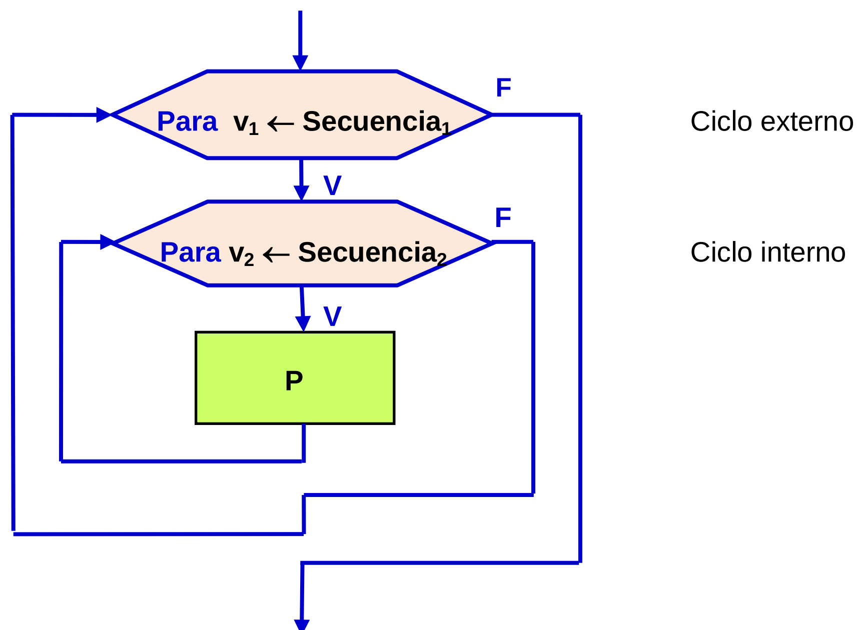
5.10.3 Ciclos anidados

Algunos algoritmos requieren ciclos dentro de otros ciclos. La regla establece que para cada instancia del ciclo externo, el ciclo interno se realiza completamente.

Una representación gráfica de dos ciclos **"while"** anidados:



Una representación gráfica de dos ciclos **"for"** anidados:



P representa el bloque de instrucciones que se repite

Ejemplo. Liste todas las parejas de números con valores enteros del 1 al 3

Solución

Variables

a,b: contendrán los enteros 1, 2, 3

Programa

```
#Parejas de números
for a in [1,2,3]:
    for b in [1,2,3]:
        print(a,b)
```

Prueba del programa

```
>>>
 1 1
 1 2
 1 3
 2 1
 2 2
 2 3
 3 1
 3 2
 3 3
```

Otra solución para el ejemplo anterior

```
#Parejas de números
for a in range (1,4):
    for b in range (1,4):
        print(a,b)
```

Prueba del programa

```
>>>
 1 1
 1 2
 1 3
 2 1
 2 2
 2 3
 3 1
 3 2
 3 3
```

Se producen los mismos resultados. Note que el extremo final del rango no se incluye.

Ejemplo. Liste todas las parejas de números con valores enteros del 1 al 3 pero sin que hayan parejas repetidas

Solución

Variables

a: contendrá los enteros 1, 2, 3
b: contendrá los enteros desde el valor de **a** hasta 3

Se usará un rango para el ciclo interno que se inicie con el valor del rango del ciclo externo. Esto evita que el ciclo interno use valores anteriores que ya fueron considerados

Programa

```
#Parejas de números sin repeticiones
for a in range (1,4):
    for b in range (a,4):
        print(a,b)
```

Prueba del programa

```
>>>
1 1
1 2
1 3
2 2
2 3
3 3
>>>
```

Ejemplo. Para cada mes muestre una lista numerada de las cuatro semanas.

Solución

Variables

m: contendrá el nombre del mes
s: contendrá los números de las semanas

Programa

```
for m in ['Ene','Feb','Mar','Abr','May','Jun','Jul',
          'Ago','Sep','Oct','Nov','Dic']:
    print('Mes: ',m)
    for s in range(1,5):
        print(' Semana: ',s)
```

Note el encolumnamiento de las instrucciones dentro de cada ciclo

Prueba del programa

```
>>>
Mes:  Ene
 Semana:  1
 Semana:  2
 Semana:  3
 Semana:  4
Mes:  Feb
 Semana:  1
 Semana:  2
 Semana:  3
 Semana:  4
Mes:  Mar
 Semana:  1
 Semana:  2
 Semana:  3
 Semana:  4
Mes:  Abr
 Semana:  1
 Semana:  2
 Semana:  3
 Semana:  4
Mes:  May
 Semana:  1
 Semana:  2
 Semana:  3
 Semana:  4
Mes:  Jun
 Semana:  1
 Semana:  2
 Semana:  3
 Semana:  4
Mes:  Jul
 Semana:  1
 Semana:  2
 Semana:  3
 Semana:  4
Mes:  Ago
 Semana:  1
 Semana:  2
 Semana:  3
 Semana:  4
 . . .
```

etc.

Ejemplo. Dado un entero positivo encuentre sus factores primos

Los factores primos son los números primos que conforman el mayor conjunto de divisores enteros positivos de un número, tales que su producto es igual al número dado.
Ejemplo: si el dato es 120, sus factores primos son: 2, 2, 2, 3, 5 pues su producto es 120

Solución

Variables

- **x:** Dato entero positivo
- **n:** Cada número natural se probará como posible divisor de **x**

Se usará un ciclo para probar cada número natural **n** comenzando en **2** hasta llegar a **x**. Si **n** es un divisor de **x** se mostrará este divisor y se reducirá el valor de **x** dividiéndolo para este divisor **n**. Esto se realizará en un ciclo pues el número **x** puede ser divisible para **n** más de una vez.

Programa

```
#Factores primos de un número entero
x = int(input('Ingrese el dato: '))
n = 2
while n<=x:
      while x%n == 0:               #Probar el divisor n
            print('Divisor: ', n)
            x=x/n                   #Reducir el dato x
      n=n+1
```

Prueba del programa

```
>>>
Ingrese el dato: 120
Divisor:  2
Divisor:  2
Divisor:  2
Divisor:  3
Divisor:  5
```

Este ejemplo muestra algunos aspecto de interés. El uso de un ciclo **while** dentro de otro ciclo **while.** En estos casos, el ciclo interno se realiza completamente antes de salir nuevamente al ciclo externo.

El encolumnamiento de las instrucciones define cuales pertenecen a cada ciclo.

Debido a que este algoritmo es algo más complejo que los ejemplos anteriores se ha descrito en forma abreviada la idea propuesta para resolver este problema antes de construir el programa. También se describe una prueba manual de la idea propuesta.

En el ejemplo también se observa la escritura de comentarios al inicio y también al final de algunas instrucciones.

Prueba manual del programa

		Salida	x	n
			120	2
Ciclo externo	**Ciclo interno**			
2≤120				
	120 % 2 = 0	**2**	60	
	60 % 2 = 0	**2**	30	
	30 % 2 = 0	**2**	15	
	15 % 2 ≠ 0			
				3
3≤15				
	15 % 3 = 0	**3**	5	
	5 % 3 ≠ 0			
				4
4≤5				
	5 % 4 ≠ 0			
				5
5≤5				
	5 % 5 = 0	**5**	1	
				6
6 no es ≤ 1				

Ejemplo. Liste todas las ternas **(a, b, c)** de números enteros entre **1** y **20** que cumplen la propiedad Pitagórica: $a^2 + b^2 = c^2$

Solución

Variables

a, b, c: números enteros entre **1** y **20**

Programa

```
#Ternas Pitagóricas
for a in range (1,21):
    for b in range (1,21):
        for c in range (1,21):
            if a**2+b**2==c**2:
                print(a,b,c)
```

Prueba del programa

```
>>>
3 4 5
4 3 5
5 12 13
6 8 10
8 6 10
```

```
8 15 17
9 12 15
12 5 13
12 9 15
12 16 20
15 8 17
16 12 20
>>>
```

La lista de resultados incluye ternas repetidas

Ejemplo. Liste todas las ternas **(a, b, c)** de números enteros entre 1 y 20 que cumplen la propiedad Pitagórica: $a^2 + b^2 = c^2$ pero sin incluir ternas repetidas

Solución

Variables

a, b, c: números enteros entre **1** y **20**

Es una modificación del ejemplo anterior. El rango de cada ciclo interno comienza en el valor del rango del ciclo externo para que no se repitan valores que ya fueron considerados antes.

Programa

```
#Ternas Pitagóricas
for a in range (1,21):
    for b in range (a,21):
        for c in range (b,21):
            if a**2+b**2==c**2:
                print(a,b,c)
```

Prueba del programa

```
>>>
3 4 5
5 12 13
6 8 10
8 15 17
9 12 15
12 16 20
>>>
```

Pregunta: ¿Cuantas repeticiones se realizan en total en cada uno de los dos últimos algoritmos?

Ejemplo. Determine si la proposición lógica: **(a∧b)⇒(a∨c)** es una tautología.

Para que una expresión lógica sea una tautología, el resultado debe ser **verdadero** para todas las combinaciones de los valores lógicos de las variables.

Solución

Variables

a,b,c: cada variable tomará los valores lógicos: **True** y **False**

El lenguaje Python no tiene un operador para el enunciado ⇒ pero se puede usar una equivalencia lógica: **p⇒q ≡ ⌉p∨q.**

Con esta equivalencia, la expresión **(a∧b)⇒(a∨c)** se transforma a: **⌉((a∧b)) ∨ (a∨c)**

Programa

```
#Verificar tautología
for a in [True,False]:
    for b in [True,False]:
        for c in [True,False]:
            p=not((a and b)) or (a or c)
            print(a,b,c,p)
```

Prueba del programa

```
>>>
True True True True
True True False True
True False True True
True False False True
False True True True
False True False True
False False True True
False False False True
>>>
```

El resultado muestra que la expresión lógica **si** es una **tautología** pues la cuarta columna que contiene el resultado para cada una de las combinaciones de valores lógicos de las variables, es en cada caso **verdadero**.

Ejemplo. Se dice que dos números son "amigables" si el primero es la suma de los divisores del segundo y viceversa. Escriba un programa que lea dos números y determine si so "amigables".

Prueba: Los números 220 y 284 son "amigables" pues se cumple que:
Los divisores de 220 son: {1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110} y la suma es 284.
Los divisores de 284 son: {1, 2, 4, 71, 142} y la suma es 220.

Solución

Variables

- **a,b**: Datos (números enteros positivos)
- **n:** Cada posible divisor
- **s**: Suma de los divisores menores que **a**
- **t**: Suma de los divisores menores que **b**

La suma de los divisores de **a** se compara con la suma de los divisores de **b**

Programa

```
# Números amigables
a=int(input('Primer número: '))
b=int(input('Segundo número: '))
s=0
for n in range(1,a):
    if a%n==0:
        s=s+n                # Suma los divisores de  a
t=0
for n in range(1,b):
    if b%n==0:
        t=t+n                # Suma los divisores de  b
if s==b and t==a:
    print('Son números amigables')
else:
    print('No son números amigables')
```

Prueba del programa

```
>>>
Primer número: 230
Segundo número: 540
No son números amigabless

>>>
Primer número: 220
Segundo número: 284
Son números amigables
>>>
```

Ejemplo. Escriba un programa que busque todas las parejas de números “amigables” entre los numeros naturales menores a 500.

Solución

Variables

a, b: Variables que toman cada valor entero entre 1 y 500
n: Cada número natural
s: Suma de los divisores menores que **a**
t: Suma de los divisores menores que **b**

Cada suma de los divisores de **a** se compara con cada suma de los divisores de **b**

Programa

```
# Números amigables
for a in range(1,500):
    s=0
    for n in range(1,a):
        if a%n==0:
            s=s+n                # Suma los divisores de  a
    for b in range(1,500):       # Para cada suma de a se calcula
        t=0                      # la suma de los divisores de  b
        for n in range(1,b):
            if b%n==0:
                t=t+n            # Suma los divisores de  b
        if s==b and t==a and a!=b:     # Compara las sumas
            print(a,b)
```

Prueba del programa

```
>>>
220 284
284 220
>>>
```

Los resultados muestran que solamente hay una pareja de números “amigables” entre 1 y 500. Note que verificar manualmente este resultado sería muy laborioso.

5.10.4 La instrucción break

Esta instrucción se utiliza para interrumpir las repeticiones de un ciclo y salir sin completar la cantidad de iteraciones que estaba prevista.

Ejemplo. Simular lanzamientos de un dado. Determinar la cantidad de intentos realizados hasta que salga el **5**.

Solución usando un ciclo for y la instrucción break

Variables

n: cantidad máxima de intentos que se realizarán
x: número aleatorio entre 1 y 6

Programa

```
# Simular lanzamientos de un dado
from random import*
n=int(input('Cantidad máxima de lanzamientos: '))
for i in range(n):
    x=randint(1,6)
    print(x)
    if x==5:
        print('Lanzamiento en el cual salió el 5: ',i+1)
        break
```

Prueba del programa

```
>>>
Cantidad máxima de lanzamientos:  20
2
2
1
2
1
2
5
Lanzamiento en el cual salió el 5:  7
>>>
```

En este ejemplo la solución no luce muy natural pues la salida normal del ciclo **for** ocurre cuando se agotan los valores del rango o lista. La instrucción **break** permite salir del ciclo antes de agotar la lista de valores.

Esta solución no garantiza que se obtenga el número **5** dentro de la cantidad máxima de lanzamientos especificada.

La manera más natural de construir algoritmos cuando no se conoce cuantos ciclos deben realizarse hasta llegar a la solución, es la instrucción de repetición **while** como se muestra en la siguiente solución.

Ejemplo. Simular lanzamientos de un dado. Determinar la cantidad de lanzamientos hasta que salga el **5**.

Solución usando la estructura while

La estructura **while** se puede usar para mantener la repetición mientras se cumple una condición, y no se necesita usar la instrucción **break**

Variables

n: cantidad de intentos
c: conteo de repeticiones
x: número aleatorio entre 1 y 6

Programa

```
# Simular lanzamientos de un dado
from random import*
c=0
x=0
while x!=5:
    x=randint(1,6)
    print(x)
    c=c+1
print('Lanzamiento en el cual salió el 5: ',c)
```

Prueba del programa

```
>>>
3
2
2
4
1
6
1
5
Lanzamiento en el cual salió el 5:  8
>>>
```

El valor inicial **x = 0** permite ingresar al ciclo **while.** Dentro del ciclo se genera la secuencia de números aleatorios hasta que se cumple la condición de salida.

Como se verá en secciones posteriores, la programación de algunos algoritmos puede necesitar el uso de la instrucción **break** para salir de ciclos. De todas maneras esta instrucción debe usarse con precaución para mantener la claridad de la codificación de los programas.

Ejemplo. Diseñe un programa para el juego de adivinar un entero generado al azar. Incluya un mensaje de ayuda y un conteo de intentos.

Solución

Variables

x: número aleatorio entero
n: número ingresado
i: conteo de intentos

Programa

Se usa un ciclo **while** con la condición **True**. Esto mantiene el ciclo activo hasta que adivine el número. La instrucción **break** forza la finalización del ciclo

```
from random import*
x=randint(1,100)
i=0
while True:
    i=i+1
    n=int(input('Adivine el número: '))
    if n==x:
        print('Adivinó en ',i,' intentos')
        break
    else:
        if n<x:
            print('Muy pequeño')
        else:
            print('Muy grande')
```

Prueba del programa

```
>>>
Adivine el número: 50
Muy grande
Adivine el número: 25
Muy pequeño
Adivine el número: 42
Muy pequeño
Adivine el número: 48
Muy grande
Adivine el número: 46
Muy grande
Adivine el número: 45
Adivinó en  6  intentos
>>>
```

Interrupción de las iteraciones de un ciclo doble

La instrucción **break** interrumpe las iteraciones de un ciclo. Para salir de un ciclo doble se puede usar un artificio como se muestra en el siguiente programa. Suponer que se desea salir de ambos ciclos la primera vez que el valor asignado a la variable **n** sea divisible por 7

```
salida = False
for i in range(1,10):
    for j in range(1,10):
        n=2*i**3+3*j**2
        if n%7==0:
            print(i,j,n)
            salida = True
            break            # Sale del ciclo interno
    if salida:
        break                # Sale del ciclo externo
```

Al activarse la instrucción **break** en el ciclo interno se asigna un valor lógico a la variable **salida** y sale del ciclo interno. El valor de la variable **salida** es usado en el ciclo externo para salir también del ciclo externo.

Prueba del programa

```
>>>
1 2 14
>>>
```

5.10.5 La instrucción continue

La instrucción **continue** se usa en los ciclos para regresar al comienzo del ciclo ignorando todas las instrucciones restantes en la iteración actual.

Ejemplo. El siguiente programa imprime los caracteres de una palabra ignorando los espacios en blanco.

```
#Uso de continue
x='Lu  ne s'
for c in x:
    if c==' ':
        continue
    print(c)
```

Prueba del programa

```
>>>
L
u
n
e
s
```

5.10.6 La instrucción exit

Esta instrucción se encuentra en el módulo **sys**. Se usa para forzar la finalización de un programa antes de la salida normal.

```
from sys import*
. . .
. . .
. . .
exit()                              Finalizará la ejecución del programa
```

Una de las normas de la **programación estructurada** es evitar instrucciones que alteran el flujo establecido en las estructuras de control de las **decisiones y ciclos.** Esto ocurre con las instrucciones **break, continue** y **exit** por lo tanto, el uso de estos recursos que alteran el flujo normal debe restringirse a los casos en los cuales sea necesario.

5.10.7 La instrucción pass

La instrucción **pass** representa una operación nula. Cuando es ejecutada nada ocurre. Se la usa como un relleno en algún lugar del programa en donde deben escribirse instrucciones, pero al no estar listas se escribe esta instrucción.

Ejemplo.

```
if m in [1,3,5,7,8,10,12]:
    print('Mes de 31 días')
else:
    pass         # Reemplaza al código que aun no se escribe
```

5.10.8 El objeto None

Este nombre especial se usa para crear variables pero sin especificar aún ni el valor ni su tipo

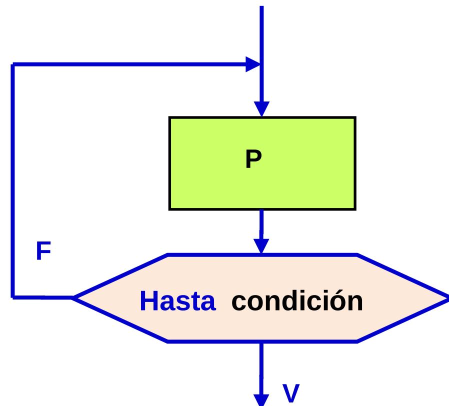
Ejemplo.

```
>>> x=None
```

5.10.9 Ejecución repetida de un bloque mediante una condición al final

Algunos programadores prefieren definir ciclos con la condición al final. Esta estructura se usa para **repetir** un bloque **hasta** que se cumpla alguna condición. Algorítmicamente se la puede representar con el siguiente gráfico:

Representación gráfica



Al entrar a esta estructura, se ejecutan las instrucciones en el bloque y después se chequea el valor de la condición. Si la condición tiene el valor verdadero, termina el ciclo y la ejecución continua abajo, caso contrario, nuevamente se repite el bloque. En esta estructura algorítmica, el bloque de instrucciónes **se repite al menos una vez**.

Seudo lenguaje

Repita

P

Hasta condición

Lenguaje Python

El lenguaje Python no dispone directamente de esta estructura de control pero se la puede traducir usando la instrucción **while** y la instrucción **break** de la siguiente manera:

```
while True:
     instrucción en el bloque P
     instrucción en el bloque P
     . . .
     instrucción en el bloque P
     if condición: break
```

El ciclo **while** permanece repitiendo el bloque de instrucciones hasta que se cumpla la condición. En este caso, se activa la instrucción **break** y el ciclo termina

Ejemplo. Simular lanzamientos de un dado. Determinar la cantidad de lanzamientos hasta que salga el **5**.

Solución con la estructura de repetición condicionada al final

```
from random import*
c=0
while True:
    x=randint(1,6)
    print(x)
    c=c+1
    if x==5: break
print('Lanzamiento en el cual salió el 5: ',c)
```

Prueba del programa

```
>>>
2
1
4
3
2
4
5
Lanzamiento en el cual salió el 5:  7
>>>
```

5.11 Introducción a validación de datos

En todos los ejemplos anteriores se ha supuesto que los datos que ingresan no tienen algún error. En la realidad al usar un programa es frecuente que se introduzcan datos incorrectos, por lo que el programa debe realizar alguna validación para evitar que ingresen estos datos y se produzcan resultados incorrectos o que adicionalmente, el programa tenga una interrupción inesperada.

Ejemplo. Dado un número entero, determine cuantas cifras tiene.

Solución

Variables

n: dato (entero positivo)
c: cantidad de cifras de **n**

El número **n** es reducido dividiéndolo sucesivamente para 10 en un ciclo. La varible **c** se usa para determinar cuantas veces se realizó la reducción.

Programa

```
#Conteo de las cifras de un número
n=int(input('Ingrese un entero positivo:  '))
c=0
while n>0:
    n=n//10
    c=c+1
print('cantidad de cifras: ',c)
```

Prueba del programa

```
>>>
Ingrese un entero positivo:  4578321
cantidad de cifras:  7
>>>
Ingrese un entero positivo:  -784
cantidad de cifras:  0
>>>
Ingrese un entero positivo:  45.2
ValueError: invalid literal for int() with base 10: '45.2'
>>>
```

Si el dato ingresado es negativo, el resultado será incorrecto. Si el dato tiene decimales, o tiene caracteres que no son numéricos se producirá un error de tipo, el cual interrumpe la ejecución del programa.

En la siguiente sección se propone una solución para este importante tema de la validación de datos.

5.11.1 Control de excepciones

Al probar un programa se pueden producir errores de ejecución. En este caso Python emite un **error** o excepción y se detiene la ejecución del programa. El **error** producido aparece en la ventana principal mediante un mensaje. Se muestran algunos casos:

Ejemplos.

```
>>> x=1/0
ZeroDivisionError: division by zero                    (división para cero)

>>> x=2*t+1
NameError: name 't' is not defined                     (si t no está definida)

>>> x=int(input('Ingrese un entero: '))
Ingrese un entero: 4.5
ValueError: invalid literal for int()                  (el dato no es un entero)

>>> from funciones import*
ImportError: No module named 'funciones'               (módulo no encontrado)
```

El mensaje en color azul que aparece en la pantalla es el error o excepción. Si un programa tuviera estos errores, la ejecución se detendría.

Para tener control sobre esta condición de error, el lenguaje Python dispone de una instrucción para manejo de excepciones, con la siguiente sintaxis:

```
try:
    Instrucciones en las que se desea detectar excepciones
except error:
    Instrucciones con acciones que deseamos realizar si hay una excepción
```

Si al ejecutar todas las instrucciones incluidas en **try** no se detecta la excepción o **error** especificado en **except** la ejecución se realiza en forma normal y continúa despues de la instrucción **try-except**.

Por otra parte, si al ejecutar las instrucciones incluidas en **try** ocurre el **error** o excepción especificado en **except**, entonces se realizan las instrucciones asociadas a **except** y después prosigue la ejecución.

Esta instrucción permite al programador tener control sobre los errores en la ejecución y la posibilidad de escribir instrucciones para describir acciones en caso de que se produzcan estos eventos. Si no se usa está instruccion, la ejecución del programa se detendría.

Si no se especifica cual es el **error** que se desea detectar, entonces se realizan las instrucciones asociadas a **except** al producirse cualquier exepción y después prosigue la ejecución.

Ejemplo. Dado un número entero positivo, determine cuantas cifras tiene. Valide que el dato sea entero.

Solución

Variables

n: dato entero y positivo
c: cantidad de cifras de **n**
num: variable que definimos para registrar si hubo error en el dato
num = **1**: dato correcto
num = **0**: dato incorrecto

Se usará la instrucción **try-except** para validar el dato.
Si el dato ingresado es erróneo se lo anotará mediante la asignación: **num = 0**

Programa

```
#Conteo de las cifras de un número
num=1
try:
    n=int(input('Ingrese un entero positivo:  '))
except ValueError:
    num=0
if num==1:
    n=abs(n)                  #Elimina el signo
    c=0
    while n>0:                #Conteo de cifras
        n=n//10
        c=c+1
    print('cantidad de cifras: ',c)
else:
    print('Error en el dato')
```

Prueba del programa

```
>>>
Ingrese un entero positivo:  45.7
Error en el dato

>>>
Ingrese un entero positivo:  abc
Error en el dato

>>>
Ingrese un entero positivo:  87462
cantidad de cifras:  5
```

Una mejor estrategia para validación consiste en ofrecer al usuario la posibilidad de corregir e ingresar nuevamente el dato, sin salir del programa.

La siguiente solución para validar el dato del ejemplo anterior usa la estrategia indicada y debería tenerse como referencia para aplicaciones en las que se necesiten realizar validaciones.

Solución

```
#Conteo de las cifras de un número con validación de datos
while True:
    try:
        n=int(input('Ingrese un entero positivo:  '))
    except ValueError:
        continue                #Regresa al inicio del ciclo
    n=abs(n)                    #Elimina el signo
    c=0
    while n>0:
        n=n//10
        c=c+1
    break                       #Finaliza el ciclo
print('Cantidad de cifras: ',c)
```

Prueba del programa

```
>>>
Ingrese un entero positivo:  abc
Ingrese un entero positivo:  7.259
Ingrese un entero positivo:  4357129
Cantidad de cifras:  7
```

Observe con cuidado el uso de las instrucciones **try–except**, **break**, **continue** y el encolumnamiento de las instrucciones. Se sugiere elaborar una representación gráfica del procedimiento utilizado.

El ciclo **while** permanece hasta que se ingrese un dato correcto y se pueda calcular la solución y salir del ciclo.

NOTA. Al realizar pruebas con un módulo o librería que está siendo modificada, es necesario usar la opción **Restart Shell** del menú de la ventana interactiva e importar nuevamente el módulo o librería cada vez que sea modificada para realizar pruebas desde un programa o desde la ventana interactiva.

La función **type** puede usarse para determinar el tipo de un objeto: **int**, **str**, etc.

Ejemplos.

```
>>> x=35
>>> print(type(x))
<class 'int'>
>>> x='Prueba'
>>> print(type(x))
<class 'str'>
```

Se pueden usar para verificar tipos en la ventana interactiva o en programas

Ejemplos

```
>>> x=35
>>> y='Prueba'
>>> if type(x)!=type(y):
      print('Tipos diferentes')

Tipos diferentes
```

5.12 Ejercicios de programación con ciclos

Para cada ejercicio escriba y pruebe un programa en la ventana de edición de Python.

1.- Calcule el promedio, el menor valor y el mayor valor de los pesos de **n** paquetes en una bodega. Estos datos ingresan uno a la vez dentro de un ciclo. **n** es un dato ingresado al inicio.

2.- Clasifique los pesos de los **n** objetos de una bodega en tres grupos: menor a 10 Kg., entre 10 y 20 Kg., mas de 20 Kg. Los datos ingresan uno a la vez en un ciclo.

3. Determine la cantidad de términos que deben sumarse de la serie $1^2 + 2^2 + 3^3 + 4^4 + ...$ para que el valor de la suma sea mayor a un número **x** ingresado al inicio.

4.- Dado dos números enteros **a, b**, determine su máximo común divisor **m**.
Ejemplo: **a** = 36, **b** = 45 entonces **m** = 9

5. Calcule un valor aproximado para la constante **π** usando la siguiente expresión:

$$\pi/4 = 1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + 1/13 \ldots$$

La cantidad de términos es un dato que debe ser ingresado al inicio del algoritmo.

6.- Lea los votos de **n** personas. Cada voto es un número **1, 2, o 3** correspondiente a tres candidatos. Si el dato es 0 es un voto en blanco. Si es otro número es un voto nulo. Determine el total de votos de cada candidato y el total de votos blancos y nulos.

7.- Lea las coordenadas de **u, v** de la ubicación de una fábrica y las coordenada **x, y** de **n** sitios de distribución. Encuentre cual es la distancia del sitio más alejado de la fábrica

8.- Encuentre el mayor valor de la función **f(x)=sen(x)+ln(x)**, para los valores:
x = 1.0, 1.1, 1.2, 1.3, ..., 4

9.- Se tienen una lista de las coordenadas **x, y** de **n** puntos en un plano. Lea sucesivamente las coordenadas de cada punto y acumule las distancias del punto al origen. Muestre la distancia total acumulada.

10.- Determine la suma de los términos de la serie $1^3 + 2^3 + 3^3 + ... + n^3$ en donde **n** es un número natural

11.- Determine la suma de los n primeros números de la serie: **1, 1, 2, 3, 5, 8, 13, 21, ...** en la cual cada término, a partir del tercero, se obtiene sumando los dos términos anteriores

12.- El inventor del juego del ajedréz pidió a su rey que como recompensa le diera por la primera casilla 2 granos de trigo, por la segunda, 4 granos, por la tercera 8, por la cuarta 16, y así sucesivamente hasta llegar a la casilla 64. El rey aceptó. Suponga que cada Kg. de trigo consta de 20000 granos de trigo. Si cada tonelada tiene 1000 Kg. describa un algoritmo para calcular la cantidad de toneladas de trigo que se hubiesen necesitado.

En el ciclo describa la suma $2^1 + 2^2 + 2^3 + 2^4 + \ldots + 2^{64}$

13.- Una empresa compra una máquina en $20000 pagando cuotas anuales durante cinco años. La siguiente fórmula relaciona el costo de la máquina **P**, el pago anual **A**, el número de años **n** y el interés anual **r**:

$$A = P\frac{r(1+r)^n}{(1+r)^n - 1}$$

Escriba un programa que permita calcular el valor de **P** para valores de **r = 0.01,0.02, …, 0.1**

14.- Una persona tiene una lista con los precios de n artículos y dispone de una cierta cantidad de dinero. Los artículos son identificados con la numeración natural. Escriba un programa para leer estos datos y obtener los siguientes resultados
a) Muestre la identificación de los artículos que puede comprar
b) Para cada artículo cuyo precio es menor que la cantidad de dinero disponible, determine la cantidad que puede comprar.

15.- La plataforma de un transporte tiene capacidad para llevar hasta m kilos. Se tiene una lista ordenada en forma creciente con el peso de n paquetes. Determine cuantos paquetes pueden ser transportados. La elección debe hacerse comenzando con los paquetes de menor peso.

16.- En un supermercado se hace una promoción, mediante la cual el cliente obtiene un descuento dependiendo de un número de una cifra que se escoge al azar. Si el numero escogido es menor que 7 el descuento es del 5% sobre el total de la compra, si es mayor o igual a 7 el descuento es del 10%. Lea la cantidad de dinero. genere el número aleatorio y muestre cuanto dinero se le descuenta.

17.- Escriba un programa para simular la extracción de n bolas de una caja que contiene m bolas numeradas con los números naturales del 1 al m. Cada vez que se saca la bola se muestra el número y se la devuelve a la caja, por lo tanto pueden salir bolas repetidas.

18.- Escriba un programa que genere un número aleatorio con un valor entre 1 y 100 y que sea un número primo.

19.- Escriba un programa que muestre dos números aleatorios con valores enteros entre 1 y 100 tales que la suma sea un número primo.

20.- Lea un número par. Encuentre dos números al azar tales que la suma sea igual al dato dado.

21.- Lea un número par. Encuentre dos números al azar tales que sean primos y la suma sea igual al dato dado.

22.- Simule el siguiente juego entre tres ranas. Las ranas están al inicio de una pista de 20 m. En turnos cada rana realiza un salto. El salto es aleatorio y puede ser: a) Brinca y cae en el mismo lugar, b) Salta 0.5 m en la dirección correcta, c) Salta 1 m en la direccion correcta, d) Salta 0.5 m retrocediendo. Determine cual de las tres ranas llega primero a la meta.

23.- Realice la simulación de n intentos de lanzamientos de un dado con las siguientes reglas: si sale 6 gana $5. Si sale 1 gana $1. Si sale 2, 3, 4 o 5 pierde $2. Determine la cantidad acumulada al final del juego

24.- Dado un valor entero positivo **n** verifique que $\mathbf{1^3+2^3+3^3+\ldots+n^3 = (1+2+3+\ldots n)^2}$

25.- Escriba un programa que genere **n** parejas de número primos gemelos. Estos números primos tienen la propiedad que además de ser primos, la distancia entre ellos es **2**. Ejemplo. **3** y **5**, **5** y **7**, **11** y **13**, **17** y **19**, etc

26.- En un juego se debe asignar a cada persona un número mágico que se obtiene con la siguiente regla: Se suman los dígitos de la fecha de nacimiento y se suman nuevamente los dígitos del resultado hasta obtener obtener un solo dígito, como en el siguiente ejemplo:

Fecha de Nacimiento: **28/11/1989**
28 + 11 + 1989 = 2028 ⇒ 2 + 0 + 2 + 8 = 12 ⇒ 1 + 2 = 3
Entonces el número buscado es **3**

Lea tres números: dia, mes, año y muestre el número mágico correspondiente

27. Analice el siguiente programa. Escriba los resultados que se obtendrían si el dato que ingresa para **n** es **25**

```
n = int(input('Ingrese un dato:  '))
r = 0
while n>0:
    d = n%2
    n = n//2
    r = 10*r + d
    print(d, n, r)
```

28. Analice el siguiente algoritmo

```
1  Leer a, b
2  Salte a la línea 5
3  Mostrar x
4  Salte a la línea 12
5  x ← 0
6  Si a<5 salte a la línea 10
7  x ← x + a
8  Si x>b salte a la línea 11
9  Salte a la línea 7
10 x ← x + a - b
11 Salte a la línea 3
12 Fin
```

a) Construya un diagrama de flujo ordenado que sea equivalente al algoritmo dado.
b) Interprete el diagrama de flujo y codifíquelo en notación Python

29. Analice el siguiente algoritmo

```
1   Leer a, b, c
2   r ← 0
3   Si a<b ∨ c<0 salte a la línea 8
4   Si b es par salte a la línea 6
5   r ← r + c
6   b ← b - 1
7   Salte a la línea 3
8   Mostrar r
```

a) Construya un diagrama de flujo que sea equivalente al algoritmo propuesto.
b) Interprete el diagrama de flujo y codifíquelo en notación Python

30. Construya un algoritmo para resolver el siguiente problema:

En la Asamblea de un partido político hay dos posibles candidatos para inscribirlo en las elecciones de alcalde. Para elegir al candidato del partido, cada una de las **n** personas asistentes a la reunión entregan un voto. Se deben leer uno por uno los votos y determinar si alguno de los dos candidatos obtuvo más de la mitad de los votos. Este será el candidato.

31. Analice el siguiente programa que usa un ciclo **for**. Escriba un programa equivalente que produzca el mismo resultado, pero sustituyendo el ciclo **for** por un ciclo **while** Debe definir una variable para conteo de repeticiones y la condición para salir del ciclo.

```
n = int(input(‘Ingrese un dato: ‘))
s = 0
for i in range(1,n):
    s = s + i**2
print(s)
```

32. Escriba un programa con un ciclo. Dentro del ciclo se generarán tres números aleatorios con valores enteros del 1 al 10. El programa deberá terminar cuando en alguna repetición, uno de los tres números sea igual al producto de los otros dos números. Muestre los números resultantes. Muestre también la cantidad de repeticiones que se realizaron.

33. El cuadrado de cualquier número terminado en 5 se lo puede formar como el producto: (decenas)(decenas+1) + 25.

Ej. 85^2 = 10(8)10(9) + 25 = 7225
475^2 = 10(47)10(48) + 25 = 225625

Elabore un programa que verifique si se cumple esta regla con los números **5**, **10**, **15**, **20**, ..., **m**. Si no es verdad, muestre el primer número que no cumple esta regla, **m** es un dato.

34. En una empresa multinacional el número usado en la identificación de productos es un código que consta de 13 dígitos: tres para el país, cuatro para la empresa, cinco para el producto y el dígito de control para detectar errores de digitación con la siguiente regla:

Comenzado por la izquierda hasta el decimo segundo dígito, multiplique el dígito por 1 si la posición es impar y por 3 si la posición es par. Sumar los resultados de los productos y restar de la decena superior. Este último resultado debe coincidir con el dígito de control

Escriba un programa que lea un código, valide que tenga trece dígitos, calcule el dífito de control e informe el resultado.

Ejemplo. Código:: 7 7 0 2 0 0 4 0 0 3 5 0 **8**

7x1 + 7x3 + 0x1 + 2x3 + 0x1 + 0x3 + 4x1 + 0x3 + 0x1 + 3x3 + 5x1 + 0x3 = 52

Decena superior: 60

60 – 52 = **8** Coincide con el dígito de control. El código es correcto.

35. Escriba un programa que reciba un valor para n y otro para m y muestre el resultado de la siguiente suma: $\sum_{i=1}^{n}\sum_{j=1}^{m} (i^2 + j^2 + ij)$

5.13 Programas que interactúan con un menú

Los programas en los cuales el usuario interactua con el computador de manera contínua se denominan interactivos. Estas aplicaciones son útiles para diversas aplicaciones. En su forma más elemental, se usa un menú para que el usuario elija una opción. Para cada opción, el computador realizará alguna acción

Estructura básica de un programa interactivo con un menú

1) El programa muestra un menú con las opciones disponibles para el usuario
2) El usuario elige una opción
3) El programa realiza la acción solicitada

Si la interacción está en un ciclo, el programa debe incluir una opción para salir.

Ejemplo. La siguiente fórmula permite convertir un valor de temperatura entre grados **farenheit** y grados **celcius**: $c = \frac{5}{9}(f - 32)$

Escriba un programa con un **menú** para realizar la conversión en ambos casos:

Solución

Variables

c: temperatura en °C
f: temperatura en °F
x: opción seleccionada

Programa

```
#Conversión de temperaturas
while True:
    print('1) Convertir F a C')
    print('2) Convertir C a F')
    print('3) Salir')
    x=input('Elija una opción ')
    if x=='1':
        f=int(input('Ingrese grados F '))
        c=5/9*(f-32)
        print(c)
    elif x=='2':
        c=int(input('Ingrese grados C '))
        f=9/5*c+32;
        print(f)
    elif x=='3':
        print('Adiós')
        break
```

Note el uso del ciclo **while** con la condición **True** que lo mantiene activo, y la instrucción **break** para terminar el ciclo

Prueba del programa

```
>>>
1) Convertir F a C
2) Convertir C a F
3) Salir
Elija una opción 1
Ingrese grados F 220
104.44444444444444

1) Convertir F a C
2) Convertir C a F
3) Salir
Elija una opción 2
Ingrese grados C 42
107.60000000000001

1) Convertir F a C
2) Convertir C a F
3) Salir
elija una opción 3
Adiós
>>>
```

Ejemplo: La siguiente fórmula del interés compuesto relaciona la cantidad de depósitos, el valor mensual constante de cada depósito, y el interés pactado al invertir el dinero en una cuenta. Se requiere escribir un programa interactivo con un menú para su utilización

$$A = P\left[\frac{(1+x)^n - 1}{x}\right],$$ en donde

A: Valor acumulado,
P: Valor de cada depósito **mensual**
n: Cantidad de depósitos **mensuales**
x: Tasa de interés **mensual**

Solución

Se escribirá un programa con un menú para calcular alguno de los valores de: **A, P, n**

Programa

```
#Fórmula del interés compuesto
from math import *
while True:
    print('1) Valor acumulado')
    print('2) Depósito mensual')
    print('3) Número de depósitos')
    print('4) Salir')
    opc=input('Elija una opción: ')
    if opc=='1':
        p=float(input('Valor del depósito mensual: '))
        n=int(input('Número de depósitos mensuales: '))
        x=float(input('Interés anual en porcentaje: '))
        m=0.01*x/12
        a=p*((1+m)**n-1)/m
        print('Valor acumulado: ',a)
    elif opc=='2':
        a=float(input('Valor acumulado: '))
        n=int(input('Número de depósitos mensuales: '))
        x=float(input('Interés anual en porcentaje: '))
        m=0.01*x/12;
        p=a*m/((1+m)**n-1);
        print('Cuota mensual: ',p)
    elif opc=='3':
        a=float(input('Valor acumulado: '))
        p=float(input('Valor del depósito mensual: '))
        x=float(input('Interés anual en porcentaje: '))
        m=0.01*x/12;
        n=log(a*m/p+1)/log(1+m);
        print('Numero de depósitos: ',n)
    elif opc=='4':
        print('Adiós')
        break
```

Prueba del programa

```
>>>
1) Valor acumulado
2) Depósito mensual
3) Número de depósitos
4) Salir
Elija una opción: 1
Valor del depósito mensual: 200
Número de depósitos mensuales: 120
Interés anual en porcentaje: 0.04
Valor acumulado:  24047.662469781626

1) Valor acumulado
2) Depósito mensual
3) Número de depósitos
4) Salir
Elija una opción: 2
Valor acumulado: 25000
Número de depósitos mensuales: 200
Interés anual en porcentaje: 0.04
Cuota mensual:  124.58587961001214

1) Valor acumulado
2) Depósito mensual
3) Número de depósitos
4) Salir
Elija una opción: 3
Valor acumulado: 25000
Valor del depósito mensual: 200
Interés anual en porcentaje: 0.04
Numero de depósitos:  124.74238345344854

1) Valor acumulado
2) Depósito mensual
3) Número de depósitos
4) Salir
Elija una opción: 4
Adiós
>>>
```

5.13.1 Ejercicios de programación con menú

Para cada ejercicio escriba y pruebe un programa en la ventana de edición de Python.

1. La suma de los términos de una sucesión aritmética está dada por

$\mathbf{s = \frac{n}{2}(a + u)}$, en donde

s: Suma de los términos,
n: Cantidad de términos
a: Primer término de la sucesión,
u: Último término de la sucesión

Escriba un programa con un menú que permita calcular: **s, n, a, u** dados los otros 3 datos.

Menú
1) Suma de términos
2) Cantidad de términos
3) Primer término
4) Último término
5) Salir

2. El precio de una pizza depende de su tamaño según la siguiente tabla:

Tamaño	Precio
Pequeña	$5
Mediana	$8
Grande	$12

Cada ingrediente adicional cuesta $1.5. Escriba un programa interactivo con un menú para elegir el tamaño de la pizza, indicar la cantidad de ingredientes adicionales y mostrar el valor a pagar.

Menú
1) Pizza pequeña
2) Pizza mediana
3) Pizza grande
4) Salir

6 Creación de funciones

En general las funciones en los lenguajes de programación son conjuntos de instrucciones que se escriben separadamente y que realizan alguna tarea especificada. Las funciones pueden usarse directamente en la ventana interactiva, o desde programas, o integrarlas en un mòdulo especial a manera de librerìa para organizar el desarrollo de un proyecto de programación.

Esta estrategia de dividir la resolución de un problema en mòdulos y funciones es parte de la metodología denominada **Programaciòn Modular** que es importante para resolver problemas grandes o complejos.

La Programaciòn Modular facilita el diseño, construcción, prueba e integración de los componentes de un programa.

El mecanismo usual para trasmitir datos a las funciones es mediante una lista de variables que se denominan parámetros. Las funciones normalmente son diseñadas para entregar resultados.

6.1 Declaración de una función

```
def nombre(parámetros):
     instrucciones
```

nombre: Es la identificación de la función
parámetros: Son variables que reciben los datos que entran a la función.
instrucciones: Se incluyen en la función para producir resultados

Las instrucciones incluidas en la función deben estar encolumnadas como en las otras estructuras de control.

Si la función entrega resultados, debe usarse una instrucción especial para retorno de resultados:

```
return variable
```

variable: es la identificación de la variable que contiene el resultado. También se puede entregar un resultado directamente.

Una función puede entregar màs de un resultado en una lista:

```
return [variable,variable,...]
```

Una función puede definirse en la ventana interactiva o en la ventana de edición. En el segundo caso, la función debe almacenarse en un archivo. El nombre del archivo puede ser igual o diferente al nombre asignado a la función en su definiciòn. El archivo almacenado consituye un módulo. Para usar la función debe importarse el módulo que contiene a la función. Un módulo puede incluir más de una función.

Ejemplo: Escribir en la ventana interactiva la siguiente función matemática:

$$y = f(x) = 2x^2 + 1$$

Definiciòn en la ventana interactiva

```
>>>  def f(x):
         y=2*x**2 + 1
         return y
```

La función está disponible para su uso inmediato:

```
>>> f(2)
9
>>> y=2*f(3)+1
>>> y
39
```

Funciones de una línea

Si una función tiene una forma simple, puede escribirse en forma abreviada en una línea

```
>>> def f(x): return 2*x**2+1
```

Se la puede usar igual que la definición anterior

```
>>> f(2)
9
```

En el siguiente ejemplo, la función es creada en la ventana de edición y se almacena separadamente en un archivo con un nombre. Este objeto constituye un módulo. El nombre del módulo puede coincidir o puede ser diferente al nombre de la función.

Solución

Nombre de la función: **f**
Nombre del módulo: **función**

```
def f(x):
    y=2*x**2 + 1
    return y
```

Para usarla, debe importarse el módulo

```
>>> from función import f
>>> y=f(2)
>>> y
 9
```

NOTA. Los nombres en el lenguaje Python admiten **tildes** y la letra **ñ**

Las funciones pueden escribirse junto al programa que las usan. En este caso, no se requiere importar el módulo. Es una buena idea hacerlo para probar funciones nuevas que están siendo desarrolladas.

Ejemplo.

```
def f(x):
    y=2*x**2 + 1
    return y

#Programa que usa la función f
for i in range(5):
    y=f(i)
    print(i,y)
```

Prueba del programa

La ejecución del programa se activa desde la ventana de edición

```
>>>
0 1
1 3
2 9
3 19
4 33
>>>
```

Un módulo puede incluir varias funciones. Estos módulos pueden considerarse **librerías**.

Ejemplo. Defina un módulo con el nombre **funciones** para almacenar las funciones

$f(x)=2x^2+1$

$g(x)=3x^3+5$

```
def f(x):
    y=2*x**2 + 1
    return y
def g(x):
    y=3*x**3 + 5
    return y
```

Para usar las funciones en la ventana interactiva debe importar el módulo

```
>>> from funciones import*
>>> f(3)
19
>>> g(4)
197
>>>
```

Los programas que requieran usar las funciones pero que se escriben separadamente de las definiciones de las funciones, deben importar el módulo

```
#Programa que usa el módulo funciones
from funciones import*
for i in range(5):
    y=f(i)
    print(i,y)
```

Prueba del programa

```
>>>
0 1
1 3
2 9
3 19
4 33
>>>
```

Ejemplo. Defina un módulo (librería) con el nombre **geometría** que contenga funciones con algunas fórmulas de la geometría básica para calcular área y volumen. Incluya fórmulas para el círculo, sector de círculo, segmento de círculo, esfera, cilindro y cono.

```
from math import*
def circulo(r):
    s=pi*r**2
    return s

def sector(r,a):
    s=pi*r**2*a
    return s

def segmento(r,a):
    s=r**2*(pi*a-0.5*sin(a))
    return s

def esfera(r):
    s=4*pi*r**2
    v=4*pi*r**3/3
    return [s,v]

def cilindro(r,h):
    s=2*pi*r*(r+h)
    v=pi*r**2*h
    return [s,v]

def cono(r,h):
    g=sqrt(h**2+r**2)
    s=pi*r*g+pi*r**2
    v=pi*r**2*h/3
    return [s,v]
```

Para usar las funciones desarrolladas, debe importarse el módulo en la ventana interactiva o en algún programa.

Ejemplo. Usar en la ventana interactiva el módulo **geometría** con las funciones

```
>>> from geometria import*
>>> circulo(2)
12.566370614359172
>>> s=circulo(2)
>>> s
12.566370614359172
>>> cilindro(4,5)
(226.1946710584651, 251.32741228718345)
>>> [s,v]=cilindro(4,5)
>>> s
226.1946710584651
>>> v
251.32741228718345
```

Nota: Los nombres de las variables o parámetros que se escriben para llamar a una función pueden ser diferentes respecto a los nombres usados al definir la función.

6.2 Parámetros empaquetados

Los parámetros se pueden enviar a una función **empaquetados** en una lista. En este caso, el nombre de la lista debe estar precedido con un asterisco.

Ejemplo

```
>>> from geometria import*
>>> d=[4,5]
>>> [s,v]=cilindro(*d)
>>> s
226.1946710584651
>>> v
251.32741228718345
```

Los datos van empaquetados en la lista **d**

6.3 Parámetros por omisión

Es posible asignar valores a los parámetros para el caso que no vengan con algún valor al ser llamada la función. Esto significa que una función puede ser llamada con menos parámetros que los que se especifican en la definición.

Ejemplo

```
def fun(a,b=0):
```

El segundo parámetro se define por omisión

Al llamarla con `fun(3,5)` se asigna **3** al parámetro **a** y **5** al parámetro **b**
Al llamarla con `fun(3)` se asigna **3** al parámetro **a** y **0** al parámetro **b**

Ejemplo. Escriba una función que reciba un número y determine si es un número primo. El resultado que entrega la función será un valor lógico según corresponda.

Junto a la función escriba un programa que liste los números primos existentes en un rango especificado

Solución

Si el programa se escribe junto con la función, primero debe escribirse la función y después el programa. En este caso, el programa no necesita importar la función que lo antecede.

Variables

n: número para probar si es primo
c: conteo de números divisores en la función
a,b: rango de búsqueda de número primos

La función entrega un resultado lógico: **True** o **False**

```
#función primo
def primo(n):
    c=0
    for i in range(1,n+1):
         if n%i==0:
             c=c+1
    if c>2:
        return False
    else:
        return True

#Programa que lista primos en un rango
a=int(input('Desde: '))
b=int(input('Hasta: '))
for n in range(a,b+1):
      if primo(n):
            print('Número primo: ',n)
```

Prueba del programa

```
>>>
Desde: 20
Hasta: 40
Número primo:  23
Número primo:  29
Número primo:  31
Número primo:  37
```

Nota: Las instrucciones de prueba se las ha escrito junto a la función. Esta es una buena práctica para desarrollar funciones. Después se la puede almacenar separadamente para que pueda ser importada desde cualquier programa o desde la ventana interactiva,

Para el siguiente ejemplo se almacenará la función **primo** separadamente para que esté disponible para otros programas o para que pueda usarse desde la ventana interactiva. La función **primo** será almacenada con el mismo nombre: **primo**. Con este nombre deberá ser importada. Cuando el programa se escribe junto a la función no es necesario importar.

```
#función primo
def primo(n):
    c=0
    for i in range(1,n+1):
         if n%i==0:
             c=c+1
    if c>2:
        return False
    else:
        return True
```

Para acceder a la función, el programa debe **importar** el **módulo primo**

```
#Programa que lista primos en un rango
from primo import primo
a=int(input('Desde: '))
b=int(input('Hasta: '))
for n in range(a,b+1):
    if primo(n):
        print('Número primo: ',n)
```

Los resultados de una prueba son iguales a los del ejemplo anterior

Ejemplo. Escriba otro programa para usar la función **primo.** Encuentre parejas de números primos cuya suma sea igual a un número ingresado al inicio en el programa.

```
#Programa que encuentra parejas de primos
from primo import primo
n=int(input('Ingrese un entero positivo: '))
for i in range(n):
    for j in range(n):
        if primo(i) and primo(j) and i+j==n:
            print('Pareja de primos: ',i,j)
```

Prueba del programa

```
>>>
Ingrese un entero positivo: 16
Pareja de primos:  3 13
Pareja de primos:  5 11
Pareja de primos:  11 5
Pareja de primos:  13 3
```

Si no se desean parejas repetidas, se puede modificar el **inicio del rango** del ciclo interno:

```
for j in range(i,n):
```

6.4 Espacio de las variables de programas y funciones

Las variables creadas en los programas se pueden acceder fuera del programa, en la ventana interactiva. En cambio, las variables definidas dentro de una función son locales, es decir no se pueden acceder fuera de la función. La función normalmente solo se puede comunicar mediante parámetros y mediante los resultados entregados.

Ejemplo. Realizar una prueba nueva del programa anterior que busca los números primos en un rango especificado. El programa debe ser ejecutado desde la ventana de edición con la opción **run** o con la tecla funcional **F5**:

```
#función primo
def primo(n):
    c=0
    for i in range(1,n+1):
        if n%i==0:
            c=c+1
    if c>2:
        return False
    else:
        return True

#Programa que lista primos en un rango
a=int(input('Desde: '))
b=int(input('Hasta: '))
for n in range(a,b+1):
    if primo(n):
        print('Número primo: ',n)
```

```
>>>
Desde: 20
Hasta: 40
Número primo:  23
Número primo:  29
Número primo:  31
Número primo:  37
>>> a
20
>>> b
40
>>> c
NameError: name 'c' is not defined
```

Se puede observar que los contenidos de las variables **a**, **b** creadas en el programa son visibles en la ventana interactiva. Por otra parte, ni en el programa, ni desde la ventana interactiva se puede acceder a la variable **c** que es creada dentro de la función.

Nota: Si el programa es ejecutado desde la ventana interactiva con la instrucción **import** no se tiene acceso a las variables del espacio del programa.

6.5 Declaración de variables globales

Con la declaración

```
global v1,v2, ...
```

Es posible hacer que los programas pueden acceder a las variables $\mathbf{v_1}$, $\mathbf{v_2}$, ... creadas dentro de una función. También se tendrá acceso a estas variables desde la ventana interactiva.

Ejemplo.

```
def fun(x):
    t=2*x
    y=t+5
    return y

#Programa que usa fun
x=3
r=fun(x)
print(r)
print(t)
```

Prueba del programa

```
>>>
11
NameError: name 't' is not defined
```

Error al intentar imprimir **t**

El programa no tiene acceso a la variable **t** creada en la función y por lo tanto, local

En la siguiente prueba se declara **t** dentro de la función como variable **global**

```
def fun(x):
    global t
    t=2*x
    y=t+5
    return y

#Programa que usa fun
x=3
r=fun(x)
print(r)
print(t)
```

Prueba del programa

```
>>>
11
6
```

El programa puede imprimir el contenido de la variable **t**

6.6 Funciones sin parámetros

No es obligación que las funciones reciban o entreguen valores. En el siguiente ejemplo, una función solamente muestra una lista de mensajes.

```
def menu():
    print('1) Ingresar dato')
    print('2) Mostrar resultado')
    print('3) Salir')
```

Esta función se la almacenó en un módulo con el nombre **menues** y se la puede importar para su uso en la ventana interactiva o desde un programa:

```
>>> from menues import*
>>> menu()
1) Ingresar dato
2) Mostrar resultado
3) Salir
>>>
```

6.7 Expresiones lambda

Es una manera de definir en una línea, expresiones con parámetros. Esta definición actúa en forma similar a una función, con la siguiente sintaxis:

```
e = lambda parámetros: expresión
```

Ejemplos.

Definir un polinomio

```
>>> pol=lambda x,y: 2*x*y-x+y+1
>>> pol(2,3)
14
```

Definir una operación sobre una lista (el estudio de listas se hará en el siguiente capítulo)

```
>>> total=lambda s: 3*sum(s)+1
>>> x=[2,3,5,6]
>>> total(x)
49
```

De manera equivalente se las puede definir como funciones abreviadas de una línea:

```
>>> def pol(x,y):return 2*x*y-x+y+1
>>> def total(s):return 3*sum(s)+1
```

El uso y resultados son idénticos a las definiciones como expresiones **lambda**

6.8 Funciones recursivas

Una función puede llamarse a si misma. Estas funciones se denominan recursivas. El uso de la recursón es una técnica útil en programación para resolver algunos problemas en sustitución de los métodos iterativos con **for** o **while**. La recursión construye la solución llamándose a si misma, usualmente mediante una estructura **if**.

La recursión realiza un ciclo internamente, por lo tanto, la programación ya no requiere escribir ciclos. Sin embargo, la recursión debe usarse con precaución pues puede ser costosa en tiempo computacional especialmente si cada llamada genera más de una llamada a la misma función. Adicionalmente, los lenguajes de programación tienen un límite máximo para las llamadas recursivas.

Ejemplo. Escriba una función para sumar los cubos de los primeros **n** números naturales.

$$sc(n) = 1^3 + 2^3 + 3^3 + \ldots + (n-1)^3 + n^3$$

La manera usual de interpretar esta suma es mediante una repetición en la cual en cada ciclo se agrega un nuevo término, como se muestra a continuación:

```
def sc(n):
    r=0
    for i in range(1,n+1):
        r=r+i**3
    return r
```

Otra manera de interpretar esta suma es mediante una definición recurrente:

Si **sc(n)** es la suma: $1^3 + 2^3 + 3^3 + \ldots + (n-1)^3 + n^3$

Entonces **sc(n-1)** es: $1^3 + 2^3 + 3^3 + \ldots + (n-1)^3$

Y se puede escribir:

$$sc(n) = sc(n-1) + n^3$$

Esta definición requiere una regla para que la invocación no continue indefinidamente hacia atrás. Entonces, la definición completa es:

$$sc(n) = \begin{cases} sc(n-1) + n^3, & n > 1 \\ 1, & n = 1 \end{cases}$$

Esta regla se puede aplicar manualmente como en el siguiete caso:

$$sc(4) = sc(3) + 4^3 = sc(2) + 3^3 + 4^3 = sc(1) + 2^3 + 3^3 + 4^3 = 1 + 2^3 + 3^3 + 4^3 = 100$$

Ejemplo. Escriba una **función recursiva** para calcular la suma de los cubos de los primeros **n** números naturales usando la definición anterior.

Se la almacenará en un módulo con el mismo nombre

```
def sc(n):
    if n>1:
        r=sc(n-1)+n**3
    else:
        r=1
    return r
```

Prueba de las funciones. Ambas versiones producen el mismo resultado

```
>>> from sc import sc
>>> r=sc(4)
>>> r
100
```

NOTA: El límite de llamadas recursivas Python V3.4.1 para este ejemplo fue menor a 1000. La recursión es realizada internamente por el traductor mediante llamadas recurrentes de manera similar a su aplicación manual.

Ejemplo. Escriba una **función recursiva** para contar en cuantos intentos se puede adivinar el número de un dado.

```
from random import *
def adivina(intento):
    n=randint(1,6)
    x=int(input('Adivina el número: '))
    if x!=n:
        print('Fallaste: intenta otra vez')
        intento=intento+1
        adivina(intento)
    else:
        print('Acertaste en ',intento)
```

Prueba desde la ventana interactiva

```
>>> from adivina import adivina
>>> adivina(1)
Adivina el número: 5
Fallaste: intenta otra vez
Adivina el número: 4
Fallaste: intenta otra vez
Adivina el número: 6
Acertaste en  3
```

Se inicia la función en el intento 1

Ejemplo. Escriba una función recursiva con el nombre **fib** para obtener el **n**-ésimo término de la secuencia de Fibonnaci.

n: 1, 2, 3, 4, 5, 6, 7, 8, 9, **. . .**
fib(n): 1, 1, 2, 3, 5, 8, 13, 21, 34, **. . .**

Solución

Nombre de la función: **fib**
Nombre del módulo: **fib**

En la secuencia se puede observar que cada término de fib(n) es igual a la suma de los dos términos anteriores: **fib(n) = fib(n-2) + fib(n-1)** para **n>2**. Mientras que el resultado es **fib(n) = 1** para **n=1** y **n=2**.

```
def fib(n):
    if n==1 or n==2:
        return 1
    else:
        return fib(n-2)+fib(n-1)
```

Prueba de la función en la ventana interactiva

```
>>> from fib import fib
>>> fib(6)
8
>>> fib(2)
1
>>> fib(9)
34
>>> fib(25)
75025
```

En este ejemplo, la recursión es muy costosa computacionalmente pues cada llamada a la función produce dos nuevas llamadas recurrentes y la secuencia de llamadas se abre en ramificaciones rápidamente. Se llega muy pronto al límite máximo de llamadas recursivas permitidas. La ejecución es ineficiente pues cada llamada requiere tiempo computacional

Ejemplo.

fib(6) = fib(4) + fib(5)
= fib(2) + fib(3) + fib(3) + fib(4)
= 1 + fib(1) + fib(2) + fib(1) + fib(2) + fib(2) + fib(3)
= 1 + 1 + 1 + 1 + 1 + 1 + fib(1) + fib(2)
= 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1
= 8

La recursión debe ser usada con precaución y debe evitarse cuando la función genera más de una llamada a si misma en cada invocación, como en el ejemplo anterior:

6.9 Funciones generadoras

Una función generadora es una función especial para producir una secuencia de valores

Ejemplo. Suponer que se necesita una lista de los cubos de los números naturales

La siguiente es una función elemental típica que recibe un entero y entrega su cubo. Estas funciones se limitan a entregar **un valor**:

```
def cubo(n):
    c=n**3
    return c
```

Mediante una instrucción de repetición se puede llamar sucesivamente a la función para recibir cada resultado:

```
from cubo import cubo
for n in range(5):
    c=cubo(n)
    print(c)
```

Prueba del programa

```
>>>
0
1
8
27
64
```

Una manera diferente de enfrentar este pequeño problema es crear la lista de valores mediante un dispositivo que produce la secuencia pero entrega los valores uno a la vez. Este dispositivo se denomina **función generadora**

Una función generadora usa la palabra especial **yield** para entregar **uno por uno** los valores de la secuencia. Mediante la función especial **next** se pueden solicitar sucesivamente cada valor de la secuencia.

Ejemplo. Una función generadora para producir la secuencia de n cubos

```
def cubo(n):
    for i in range(n):
        c=i**3
        yield c
```

Activación de la función generadora de cubos en la ventana interactiva

```
>>> from cubo import cubo
>>> c=cubo(5)                  Inicia el generador
>>> print(next(c))             La función next solicita el siguiente valor
0
>>> print(next(c))
1
>>> print(next(c))
8
>>> print(next(c))
27
>>> print(next(c))
64
>>> print(next(c))

StopIteration
```

También se puede solicitar cada valor mediante una instrucción iterativa y el operador **in** que implícitamente usa **next** para obtener el siguiente valor de la secuencia.

Un programa para obtener cada valor de la secuencia creada por la función generadora cubo:

```
from cubo import cubo
for c in cubo(5):
    print(c)
```

Prueba del programa

```
>>>
0
1
8
27
64
>>>
```

Los generadores actúan de manera diferente a las funciones normales. Una función normal es llamada una sola vez y entrega los resultados. Por otra parte, la función generadora produce una secuencia pero solo entrega un valor a la vez por lo que debe ser activada sucesivamente en forma explícita con **next** o en forma implícita con el operador **in** para obtener cada uno de los siguientes valores.

6.9.1 Generadores infinitos

El concepto de función generadora permite definir **generadores infinitos**

Ejemplo. Definición de un generador infinito de cubos:

```
def cubo():
    n=0
    while True:
        c=n**3
        yield c
        n=n+1
```

Un programa que usa el generador infinito necesita interrumpir la secuencia:

```
from cubo import cubo
for c in cubo():
    print(c)
    if c>300:
        break
```

Prueba del programa

```
>>>
0
1
8
27
64
125
216
343
```

Desde la ventana interactiva

```
>>> from cubo import cubo
>>> for c in cubo():
        print(c)
0
1
8
27
64
125
216
343
512
729
. . .
```

Se necesita interrumpir la secuencia infinita presionando las teclas **Ctrl C**

Ejemplo. Generador infinito de números primos

```
def primo():
    n=0
    while True:
        n=n+1
        c=0
        for d in range(2,n):
            if n%d==0:
                c=c+1
        if c==0:
            yield n
```

```
from primo import primo
for c in primo():
    print(c)
    if c>10:
        break
```

```
>>>
1
2
3
5
7
11
```

Ejemplo. Generador infinito de la secuencia de Fibonacci

```
def fib():
    a, b = -1, 1
    while True:
        c = a + b
        yield c
        a, b = b, c
```

```
from fib import fib
c=fib()
for i in range(7):
    print(next(c))
```

```
>>>
0
1
1
2
3
5
8
```

6.9.2 Interrupción de un ciclo doble

Una función generadora permite definir un dispositivo para interrumpir de manera elegante un ciclo doble:

Ejemplo. Suponer que se desea encontrar el **primer caso** en el que los valores **i, j** cumplen la condición:

$2i^3 + 3j^2$ es divisible para **7,** para **i,j = 1,2,3,. . .,9**

En el siguiente programa, la instrucción **break** solamente interrumpe las repeticiones del ciclo en el que se ejecuta la instrucción **break**, en este caso el **ciclo interno**:

```
for i in range(1,10):
    for j in range(1,10):
        n=2*i**3+3*j**2
        if n%7==0:
            print(i,j,n)
            break
```

Prueba del programa

```
>>>
1 2 14
2 2 28
4 2 140
7 7 833
8 2 1036
9 2 1470
>>>
```

La estrategia para salir de ambos ciclos es convertirlos en un solo ciclo. Para esto se escribe una **función generadora** que produce una nueva pareja de índices **i,j** cada vez que es llamada la función. En este caso, la instrucción **break** al interrumpir un ciclo, es equivalente a interrumpir los dos ciclos incluidos en el programa anterior.

```
def rango_doble(n,m):
    for i in range(1,n):
        for j in range(1,m):
            yield [i,j]

for i,j in rango_doble(10,10):
    n=2*i**3+3*j**2
    if n%7==0:
        print(i,j,n)
        break
```

Prueba del programa

```
>>>
1 2 14
>>>
```

6.10 Sugerencias generales para programar con funciones

La realización de cada módulo, programa o función comprende cuatro etapas: analizar, diseñar, instrumentar y probar. Si todas son realizadas por una misma persona, ésta debe poner algún cuidado en la documentación pues pueden quedar vacíos al no tener que escribir los detalles que tendría que agregar para comunicarse con otra persona.

Algunas sugerencias para escribir programas y funciones:

a) Dibuje un diagrama o describa una jerarquía con los componentes que integrarán su proyecto. Planifique y documente el desarrollo de cada programa y cada función. Asígne nombres a los componentes, defina objetivos, los nombres de variables y las restricciones.

b) Al inicio de cada programa o función escriba un título y una breve descripción.

c) Cada vez que realice una modificación, guarde una copia de la versión anterior con un nombre que la identifique. Escriba unas líneas indicando cual es la modificación. Realice y documente una modificación a la vez y realice las pruebas respectivas.

d) Si es posible, compare los resultados obtenidos con resultados conocidos.

e) Realice pruebas con pequeños grupos de datos antes de tratar casos grandes.

f) Escriba separadamente cada función. Junto a la función escriba algunas líneas para realizar las pruebas necesarias de la función. Después puede eliminar las líneas de prueba. Finalmente junte en una librería común las funciones que ya han sido probadas.

g) Prefiera escribir funciones en lugar de programas. Los datos para los programas deben tener un tipo específico, mientras que los parámetros de las funciones pueden recibir datos de diferente tipo, por lo tanto no están atadas a un solo tipo de datos como ocurre en los programas.

h) Aproveche la ventana interactiva de Python para probar su codificación y desarrollar prototipos mediante algoritmos simples. En una etapa posterior puede mejorar la eficiencia usando algoritmos más elaborados.

6.11 Ejercicios con funciones

1. Escriba una función **conteo(n)** que entregue la **cantidad** de divisores enteros positivos que tiene un número entero dado n. Escriba un programa de prueba que use la función escrita para encontrar cual número entre 1 y 100 tiene más divisores enteros.

2. Escriba un programa de prueba que use la función **primo** y encuentre dos números enteros aleatorios menores que 100 tales que su suma sea también un número primo.

3. Escriba una función **perfecto(n)** que determine si un número entero dado n es un número perfecto. Un número perfecto debe ser igual a la suma de todos sus divisores enteros menores que el valor del número.

Ejemplo: 28 = 1 + 2 + 4 + 7 + 14

Escriba un programa de prueba que use la función escrita y encuentre los números perfectos entre 1 y 1000

4. Escriba una función **sumad(n)** que entregue la suma de las cifras de un número dado **n**. Con esta función escriba un programa que genere 10 números aleatorios entre 1 y 100 y encuentre cual de ellos tiene la mayor suma de sus cifras.

5. Escriba una función **cuad(n)** que determine si el cuadrado de un número natural n dado, es igual a la suma de los primeros n números impares.

Ej. $6^2 = 1+3+5+7+9+11$

Escriba un programa de prueba que ingrese un dato desde el teclado, use la función y muestre el resultado en la pantalla.

6. Escriba una función **secuencia1(n)** que entregue el n-ésimo término de la siguiente secuencia, en la cual cada término, a partir del tercero se obtiene sumando los dos anteriores: 1, 1, 2, 3, 5, 8, 13, 21, ... Escriba un programa de prueba que ingrese un dato desde el teclado use la función y muestre el resultado en la pantalla.

7. Escriba una función **secuencia2(n)** que entregue el n-ésimo término de la siguiente secuencia, en la cual cada término, a partir del cuarto se obtiene sumando los tres anteriores: 1, 1, 1, 3, 5, 9, 17, 31, 57, ... Escriba un programa de prueba que ingrese un dato desde el teclado, use la función y muestre el resultado en la pantalla.

8. Escriba una función **sim(x)** que reciba un entero y determine si es simétrico, es decir si los dígitos opuestos alrededor del centro son iguales. Escriba un programa de prueba que genere números aleatorios entre 1 y 10000 hasta obtener un número que sea simétrico

9. Escriba una función **alfin(n)** que entregue como resultado la cantidad de veces que debe lanzarse un dado hasta que salga un número n dado como parámetro. Escriba un programa de prueba que ingrese un dato desde el teclado, use la función y muestre el resultado en la pantalla.

10. Escriba una función **conteo(x)** que determine la cantidad de términos que deben sumarse de la serie: 1*2*3 + 2*3*4 + 3*4*5 + 4*5*6+....... hasta que la suma exceda a un valor x dado. Escriba un programa de prueba que genere un número aleatorio para x entre 1 y 1000, use la función y muestre el resultado en la pantalla.

11. Escriba una función **fact(n)** que reciba un numero entero n y devuelva su factorial. Escriba un programa de prueba que genere un número aleatorio entero k entre 1 y 10. Use la función y muestre la suma de los factoriales de los primeros k números naturales

12. Escriba una función **sumadiv(n)** que reciba un número entero n y devuelva la suma de sus divisores. Escriba un programa de prueba que ingrese un dato desde el teclado, use la función y muestre el resultado en la pantalla

13. Escriba un módulo con el nombre **convertir** que contenga dos funciones:

a) **[r,t]=polar(x, y)** que reciba las coordenadas cartesianas **x,y** y entregue las coordenadas polares **r, t**

b) **[x,y]=cartesiana(r, t)** que reciba las coordenadas polares **r, t** y entregue las coordenadas cartesianas **x, y**

Formulación: $\mathbf{r = \sqrt{x^2 + y^2},\ \ t = arctan(y/x), \qquad x = r\ \ cos(t),\ \ y = r\ \ sen(t)}$

14. El siguiente es un algoritmo para generar un número aleatorio entero (seudo aleatorio)

1) Dado un número entero **x**
2) Sume los cuadrados de los dígitos del número. Este resultado es el número aleatorio

a) Escriba una función **c=aleatorio(x)** que entregue el resultado producido.
b) Escriba un programa que lea un valor inicial para **x** y llame a la función **aleatorio** repetidamente, enviando como nuevo dato, el resultado que entrega la función. Determine cuantas veces hay que llamar a la función **aleatorio** hasta que el resultado sea igual a algún valor que ya salió anteriormente. Esta cantidad se denomina longitud de la secuencia aleatoria

15. La siguiente definición recursiva produce el máximo común divisor entre dos números enteros positivos. Escriba y pruebe una función con esta definición

$$\mathbf{mcd(a,b) = \begin{cases} mcd(a-b,b), & a > b \\ mcd(a,b-a), & b > a \\ a, & a = b \end{cases}}$$

16. Analice la siguiente definición recursiva para determinar cuantos dígitos tiene un número entero. Escriba y pruebe una función. Note el uso de la división entera: ***//***

$$\mathbf{nd(n) = \begin{cases} nd(\ n//10\ ), & n \geq 10 \\ 1, & n < 10 \end{cases}}$$

17. Escriba una función recursiva para obtener la cantidad de dígitos impares que contiene un número entero positivo.

18. Escriba una función recursiva para calcular combinaciones de números con la definición:

C(m,n) = C(m-1, n-1) + C(m-1, n)

Sabiendo que: **C(m, n)=1,** si **n=0** ó **n=m**

C(m, n)=m, si **n=1** ó **n=m-1**

19. Escriba una funcion generadora y un programa de prueba para obtener la secuencia de Ulam que reciba un valor inicial y entregue sucesivamente los valores de la secuencia.

7 Tipos de datos estructurados

Además de los tipos de datos básicos revisados en las secciones anteriores, Python admite agrupaciones o estructuras de datos denominadas **colecciones** que permiten agrupar datos y que pueden tener diferente tipo. Estos tipos de datos estructurados o colecciones son: **Listas**, **Tuplas**, **Cadenas de caracteres (strings)**, **Diccionarios** y **Conjuntos.** Los tipos de datos estructurados se pueden combinar y formar tipos de datos compuestos.

Este capítulo es muy importante. Para tomar una idea general de los tipos de datos estructurados se han organizado las secciones de la siguiente forma:

En las secciones **7.1, 7.2,** y **7.3** se definen las estructuras de datos **listas**, **tuplas** y **cadenas de caracteres**. Se proporcionan ejemplos usando la ventana interactiva de Python. Es fundamental que el usuario realice la práctica computacional.

En las secciones **7.4** y **7.5** se definen las estructuras de datos **diccionarios** y **conjuntos** y se describen las operaciones básicas para su manejo.

Las secciones **7.6, 7.7** y **7.8** son más extensas. Se describen operadores, métodos, funciones y programas para manejo y aplicación de los tipos de datos estructurados: **listas** y **cadenas de caracteres** y casos especiales del tipo de datos lista: **vectores** y **matrices**.

El lenguaje Python dispone además de **librerías** especiales para operar con los tipos de datos estructurados.

Las estructuras de datos son tipos de datos dinámicos, es decir que pueden extenderse o modificarse en forma interactiva y durante la ejecución de programas.

7.1 Listas

Las listas constituyen el tipo de datos estructurado más importante, versátil y común de Python.

Una lista es una colección de datos que pueden tener diferente tipo. Los datos se escriben entre **corchetes**, separados por comas.

```
[dato, dato, dato, ..., dato]
```

Una propiedad fundamental de las listas es que los componentes de la lista **se pueden modificar** después de haberlas creado.

Las celdas son numeradas desde **cero**. El **primer** componente o primera celda, tiene índice **0**. El **segundo** componente, o segunda celda, tiene índice **1**, etc.

Se puede acceder a los componentes de una lista mediante un índice entre corchetes

Se puede acceder a varios elementos o componentes mediante un **rango** para el índice.

El rango **no** incluye el extremo derecho especificado

Ejemplos. Descipción de una lista con 5 componentes de tipo diferente

```
>>> x = ['abc',  73,  5.28,  'rs',  5]
```

La representación de la lista en celdas de memoria numeradas desde **cero**:

'abc'	73	5.28	'rs'	5
0	1	2	3	4

```
>>> x[0]
'abc'
```
Componente **0** (ubicado en la celda **0**)
(es el primer componente o primera celda)

```
>>> x[2]
5.28
```
Componente **2** (ubicado en la celda **2**)
(es el tercer componente o tercera celda)

```
>>> x[1:4]
[73, 5.28, 'rs']
```
Componentes desde **1** hasta el **3** (celdas **1** a **3**)
(el rango **no** incluye el extremo final)

```
>>> x[2:]
[5.28, 'rs', 5]
```
Componentes **2** hasta el final (celda **2** hasta el final)
los resultados también son **listas**

```
>>> x[:4]
['abc', 73, 5.28, 'rs']
```
Componenetes desde el primero hasta el **3**
(el rango **no** incluye el extremo final)

```
>>> x[-1]
5
```
Es el último componente (componente **4**)

```
>>> x[-2]
'rs'
```
Penúltimo componente (componente **3**)

```
>>> x[-3]
5.28

>>> x[-4]
73

>>> x[:-3]
['abc', 73]
```
Componentes desde el primero hasta el **-4**
(el rango **no** incluye el extremo final)

```
>>> x[-2:]
['rs', 5]
```
Componentes desde el penúltimo hasta el final

```
>>> x[:-1]
['abc', 73, 5.28, 'rs']
```
Todos los componentes menos el último
(el rango **no** incluye el extremo final)

```
>>> x[1]=45
>>> x
['abc', 45, 5.28, 'rs', 5]
```
Los componentes de una lista se pueden modificar

La lista fue modificada

```
>>> x[1:2]
[45]                    El resultado es una lista
>>> x[1]
45                      El resultado es un elemento

>>> x[6]                El índice fuera de rango produce un error

IndexError: list index out of range

>>> x=[2,4,5,3,7,6,8,9,1]
>>> x[0:-1:2]           Un tercer índice indica el incremento
[2, 5, 7, 8]
>>> x[::-1]             El incremento -1 se puede usar para invertir
[1, 9, 8, 6, 7, 3, 5, 4, 2]
```

Lista con componentes de tipo lista

```
>>> x=[123,'Algebra',[50,70],5,73.25]
>>> x[0]
123

>>> x[1]
'Algebra'

>>> x[2]
[50, 70]

>>> x[2][1]
70

>>> x[2][1]=75
>>> x
[123, 'Algebra', [50, 75], 5, 73.25]

>>> t=[[123,'María',50.2],[234,'Juan',24.2]]
>>> t[1]
[234, 'Juan', 24.2]
>>> t[1][2]
24.2
```

Listas organizadas en más niveles

```
>>> x=[[[23,45],['abc',42]],35]
>>> x[0]
[[23, 45], ['abc', 42]]
>>> x[1]
35
>>> x[0][0]
[23, 45]
```

```
>>> x[0][0][1]
45
>>> x[0][1][0]
'abc'
>>> x[0][1][0]='rstu'
>>> x
[[[23, 45], ['rstu', 42]], 35]
```

Definición de una lista vacía

```
>>> x=[]
```

La lista x no contiene elementos

Vectores

Un vector o arreglo de una dimensión es una lista con datos del mismo tipo, usualmente numérico. En una sección posterior se revisarán las operaciones, funciones y aplicaciones con vectores, estructuras fundamentales en matemáticas e ingeniería.

Ejemplo.

```
>>> v=[32, 74, 92, 25, 81]

>>> v[0]
32
>>> v[2]
92
>>> v[1:4]
[74, 92, 25]
```

Matrices

Una matriz o arreglo de dos dimensiones es una lista cuyos elementos son listas que tienen la misma longitud y contienen elementos del mismo tipo, usualmente numérico.

Estos objetos, igual que los vectores, son fundamentales en aplicaciones matemáticas y de ingeniería. En otra sección se revisarán las operaciones, funciones y aplicaciones de matrices y la librería **NumPy** que facilita el manejo de vectores y matrices.

Ejemplos.

```
>>> a=[[23,45,63],[72,81,91],[56,64,37],[34,75,26]]
```

Su representación conceptual es un cuadro en dos dimensiones.
Las filas son horizontales y las columnas son verticales numeradas desde cero

0	**23**	**45**	**63**
1	**72**	**81**	**91**
2	**56**	**64**	**37**
3	**34**	**75**	**26**
	0	**1**	**2**

```
>>> a
[[23, 45, 63], [72, 81, 91], [56, 64, 37], [34, 75, 26]]
```

```
>>> a[1]
[72, 81, 91]
```

Fila 1 (es la segunda fila numerada desde cero)

```
>>> a[1][2]
91
```

Componente ubicado en la fila 1, columna 2
(la numeración se inicia en cero)

7.2 Tuplas

Una tupla es una colección de datos que pueden tener diferente tipo. Los datos se escriben entre **paréntesis**, separados por comas con la siguiente sintaxis. Opcionalmente se pueden omitir los paréntesis:

```
(dato, dato, dato, ..., dato)
```

Los componentes de una tupla **no se pueden modificar** después de haber sido creados.

Las celdas son numeradas desde **cero**. El **primer** componente, o primera celda, tiene índice **0**. El **segundo** componente, o segunda celda, tiene índice **1**, etc.

Se puede acceder a los componentes de una tupla mediante un índice entre corchetes

Se puede acceder a varios elementos o componentes mediante un **rango** para el índice.

El rango **no** incluye el extremo derecho especificado

Ejemplo. Descipción de una tupla con 5 componentes de tipo diferente

```
>>> x = ('abc',  73,  5.28,  'rs',  5)
```

La representación de una tupla en celdas de memoria numeradas desde **cero**:

'abc'	73	5.28	'rs'	5
0	1	2	3	4

Son los mismos ejemplos usados en la sección anterior para describir las listas, pero ahora los resultados son **tuplas**. En la sección anterior, los resultados eran **listas**.

```
>>> x[0]
'abc'
```
Componente **0** (ubicado en la celda **0**)
(es el primer componente o primera celda)

```
>>> x[2]
5.28
```
Componente **2** (ubicado en la celda **2**)
(es el tercer componente o tercera celda)

```
>>> x[1:4]
(73, 5.28, 'rs')
```
Componentes desde **1** hasta el **3** (celdas **1** a **3**)
(el rango **no** incluye el extremo final)

```
>>> x[2:]
(5.28, 'rs', 5)
```
Componentes **2** hasta el final (celda **2** hasta el final)
Los resultados también son **tuplas**

```
>>> x[1] = 45
```
Error: no se pueden modificar los elementos de tuplas

```
TypeError: 'tuple' object does not support item assignment
```

Tupla con componentes de tipo lista

```
>>> x=(3,[6,7],8,(4,5),2)
>>> x[1]
[6, 7]

>>> x[1][1]=3
>>> x
(3, [6, 3], 8, (4, 5), 2)

>>> x[3][1]=3
```

Error: No se pueden modificar componentes de tuplas

```
TypeError: 'tuple' object does not support item assignment
```

Los paréntesis son opcionales para definir tuplas

```
>>> x = ('abc',  73,  5.28,  'rs',  5)
>>> x
('abc', 73, 5.28, 'rs', 5)
```

Se puede escribir

```
>>> x = 'abc',  73,  5.28,  'rs',  5
>>> x
('abc', 73, 5.28, 'rs', 5)
```

7.3 Cadenas de caracteres (strings)

Una cadena de caracteres o string, es una colección de caracteres que se escriben entre comillas simples o comillas dobles:

```
'caracteres'
"caracteres"
```

Las cadenas de caracteres también se pueden indexar pero los componentes de una cadena de caracteres **no se pueden modificar** mediante una asignación. Posteriormente se estudiarán funciones especiales para modificar el contenido de las cadenas de caracteres.

Las celdas son numeradas desde **cero**. El **primer** carácter, o primera celda, tiene índice **0**. El **segundo** carácter, o segunda celda, tiene índice **1**, etc.

Se puede acceder a los caracteres de una cadena mediante un indice entre corchetes.
Se puede acceder a varios caracteres o componentes mediante un **rango** para el índice.
El rango **no** incluye el extremo derecho especificado.

Ejemplos.

```
>>> x='programa'
```

La representación de una cadena en celdas de memoria numeradas desde **cero**:

'p'	'r'	'o'	'g'	'r'	'a'	'm'	'a'
0	1	2	3	4	5	6	7

```
>>> x[0]
'p'
```
el primer carácter o primera celda (es la celda **0**)

```
>>> x[:4]
'prog'
```
los caracteres desde la celda **0** hasta la celda **3**
(el rango no incluye el último)

```
>>> x[3:6]
'gra'
```
los caracteres en las celdas **3** a **5**

```
>>> x[-1]
'a'
```
el último carácter

```
>>> x[-2:]
'ma'
```
los dos últimos caracteres

```
>>> x[:-1]
'program'
```
todos los caracteres menos el último
(en el rango no se incluye el extremo final)

```
>>> x[1]= 'u'
TypeError: 'str' object does not support item assignment
```
Error: no se pueden modificar los componentes

7.4 Diccionarios

Los diccionarios son colecciones de datos con un formato especial que permite definir y acceder a los componentes únicamente mediante una **clave**. Cada componente de un diccionario es un par **clave:valor** y se escriben entre **llaves**, separados por comas con la siguiente sintaxis:

```
{clave: valor, clave: valor, clave: valor, ..., clave: valor}
```

Las claves pueden ser de diferentes tipos y los datos también pueden ser de diferentes tipos, inclusive listas o tuplas. **No se pueden modificar las claves pero si se pueden modificar los valores que están asociados a las claves.**

Los componentes de un diccionario no están en un orden específico

Para acceder a los elementos de un diccionario debe especificarse la **clave** entre corchetes.

Ejemplo. Descipción de un diccionario con clave numérica entera y valor tipo cadena.

```
>>> x={123: 'Algebra', 325: 'Física', 215: 'Química'}
```

Su representación conceptual en la memoria con celdas identificadas con la **clave**

'Algebra'	'Física'	'Química'
123	325	215

```
>>> x[123]
 'Algebra'
```
Mediante la clave se accede al valor

```
>>> x[215]
 'Química'
```

```
>>> x[123]='Matemáticas'
>>> x
 {123: 'Matemáticas', 325: 'Física', 215: 'Química'}
```
Se puede modificar el valor

Diccionarios con componentes de tipo lista

Defina un diccionario con clave numérica y valor asociado definido mediante una lista de dos componentes: nombre y edad

```
>>> x={123:['Anita',25],234:['Elena',34],456:['Carmen',45]}
>>> x
{456: ['Carmen', 45], 234: ['Elena', 34], 123: ['Anita', 25]}

>>> x[123]
['Anita', 25]
```

```
>>> x[123][0]
'Anita'

>>> x[123][1]
25

>>> x[123][0]='María'                    Se puede cambiar el nombre
>>> x
{456: ['Carmen', 45], 234: ['Elena', 34], 123: ['María', 25]}

>>> x[123][1]=27                         Se puede cambiar la edad
>>> x
{456: ['Carmen', 45], 234: ['Elena', 34], 123: ['María', 27]}
```

Eliminar elementos del diccionario dada una clave: del

```
>>> del x[234]
>>> x
{456: ['Carmen', 45], 123: ['María', 27]}
```

Listar las claves: keys

```
>>> c=x.keys()
>>> list(c)
[456, 123]
```

Detectar si el diccionario contiene una clave: in

```
>>> 123 in x
True
>>> 234 in x
False
```

Agregar elementos a un diccionario

```
>>> x[572]=['Juanita',26]
>>> x
{456: ['Carmen', 45], 123: ['María', 27], 572: ['Juanita', 26]}
```

Recorrido de los componentes de un diccionario

```
>>> for i in x:
        print(i,x[i])

456 ['Carmen', 45]
123 ['María', 27]
572 ['Juanita', 26]
```

7.5 Conjuntos

Los conjuntos se construyen como una lista de valores encerrados entre **llaves**. También se pueden definir conjuntos con la instrucción `set(c)` en donde **c** representa cualquier objeto que se pueda indexar, como tuplas, listas o cadenas de caracteres. Por definición, los componentes de un conjunto no están ordenados ni contienen elementos repetidos.

Se pueden usar los conjuntos para eliminar elementos repetidos y realizar operaciones matemáticas entre conjuntos**:**

El resultado de la definición y operación entre conjuntos es un objeto que **no** se puede indexar. Pero se lo puede convertir nuevamente a un objeto indexable.

Sean: **a, b**: conjuntos

Operación	Resultado
a & b	Intersección de conjuntos
a | b	Unión de conjuntos
a - b	Diferencia de conjuntos
a ^ b	Diferencia simétria de conjuntos

Ejemplos

```
>>> u={4,6,7,3,8,6}          Definición directa de un conjunto
>>> u
{8, 3, 4, 6, 7}

>>> r=set([7,3,8,6,9])       Definición de un conjunto desde una lista
>>> r
{8, 9, 3, 6, 7}

>>> x=[7,6,9,6,3,7,4]        Listas
>>> y=[5,7,4,8,9,4]
>>> a=set(x)                 Se convierten en conjuntos
>>> a
{9, 3, 4, 6, 7}              No tienen elementos repetidos
>>> b=set(y)
>>> b
{8, 9, 4, 5, 7}

>>> c=a&b                    Operaciones entre los conjuntos a y b
>>> c
{9, 4, 7}
>>> c=a|b
>>> c
{3, 4, 5, 6, 7, 8, 9}
>>> c=a-b
>>> c
{3, 6}
```

```
>>> c=a^b
>>> c
{3, 5, 6, 8}
```

>>> c[1] **Error**: los conjuntos no se pueden indexar

```
TypeError: 'set' object does not support indexing
```

Conversión de un conjunto a lista con la función list

```
>>> d=list(c)
>>> d
[3, 5, 6, 8]
```

La función **list** convierte el conjunto en una **lista**

```
>>> d[1]
5
```

La lista es indexable y modificable

Un ejemplo con cadenas de caracteres

```
>>> x='matematicamente'
>>> t=set(x)
>>> t
{'c', 'e', 't', 'a', 'n', 'i', 'm'}
```

Cadena, indexable pero no modificable

Conjunto sin repeticiones, no indexable

```
>>> r=list(t)
>>> r
['c', 'e', 't', 'a', 'n', 'i', 'm']
```

Lista de caracteres sin repeticiones

```
>>> r[1]='f'
>>> r
['a', 'f', 't', 'e', 'n', 'c', 'm']
```

La lista es indexable y modificable

Agregar elementos a un conjunto con la función add

```
>>> x=set([7,3,8,6,9])
>>> x
{8, 9, 3, 6, 7}
>>> x.add(5)
>>> x
{3, 5, 6, 7, 8, 9}
```

Pertenecia de un elemento en un conjunto con el operador in

```
>>> 6 in x
True
>>> 4 in x
False
```

7.6 Programación de iteraciones con tipos de datos estructurados

Las iteraciones, repeticiones o ciclos son procedimientos fundamentales para resolver problemas en computación, especialmente cuando se las usa para recorrer o iterar sobre colecciones de datos.

En esta sección se proponen algunas sugerencias para plantear mejor el uso de iteraciones con el objetivo de que la programación sea más clara y eficiente. En general, el usuario debe tratar de reducir el uso de variables y concentrarse en los objetos importantes del problema que trata de resolver.

Para desarrollar la idea anterior, supondremos que se tiene una variable **cursos** de tipo lista conteniendo los nombres de las materias que ofrece una institución educativa. Cada elemento de esta lista debe ser procesado. Por simplicidad, simplemente se lo imprimirá.

La manera clásica que usan algunos programadores para iterar sobre una lista es definiendo un índice y controlándolo en forma explícita como se muestra a continuación:

```
cursos = [. . .         ]
n=len(cursos)
i=0
while i<n:
    c=cursos[i]
    print(c)
    i=i+1
```

El índice es una variable adicional que hay que atender pero lo que realmente interesa son los componentes de la lista

Una mejor manera de iterar sobre los elementos de la lista es dejar que la instrucción **for** controle el índice. Los valores para el índice son generados con la función **range**:

```
cursos = [. . .         ]
for i in range(len(cursos)):
    c=cursos[i]
    print(c)
```

Igualmente se utiliza un índice, pero al menos ya no necesitamos controlarlo de forma explícita.

Existe una manera más clara de iterar sobre los elementos de la lista. Esta consiste en acceder directamente a los elementos, con la siguiente forma:

```
cursos = [. . .         ]
for c in cursos:
    print(c)
```

Esta repetición evita el uso de variables adicionales o índices y permite concentrar la atención en lo que realmente importa**:** los componentes de la lista.

De igual manera, suponer que se tiene una cadena de caracteres almacenada con el nombre de variable **texto**.

Una solución clásica usa un índice para acceder a cada carácter de la cadena almacenada:

```
texto = ' . . .             '
for i in range(len(texto)):
    print(texto[i])
```

Una mejor manera es acceder directamente al componente de interés**:** cada carácter. Esto evita distraer la atención en la variable adicional para el índice:

```
texto = ' . . .             '
for c in texto:
    print(c)
```

Los objetos sobre los cuales se puede **iterar** se llaman **iterables**, por ejemplo, una lista es un objeto **iterable**. El proceso de recorrer la lista se llama **iterar**.

La función **range** es el **iterador** el cual genera la secuencia de valores para iterar. La instrucción **for** es el dispositivo para iterar.

7.7 Operaciones con listas

En esta sección se establecen los instrumentos para desarrollar aplicaciones con listas. La mayoría de los ejemplos usarán **listas numéricas,** comúnmente denominados vectores o arreglos de una dimensión. Sin embargo, se pueden aplicar a listas con elementos de tipos diferentes.

7.7.1 Métodos, operadores y funciones para manejo de listas

Python dispone de dispositivos denominados Clases que contienen funciones definidas para ciertas aplicaciones. Estas funciones se denominan métodos y la metodología se denomina Programación Orientada a Objetos la cual se estudiará posteriormente.

Para manejo de listas, Python tiene una clase denominada **list**. Esta clase es residente en la librería estándar y no necesita importarse para usarla.

Al crear una variable u objeto de tipo lista se tiene acceso a las funciones o métodos definidos en la clase y se los usa con la siguiente notación:

objeto.método

Algunos métodos disponibles en la clase **list**:

Sean **x**: objeto de tipo lista
e: un elemento

Método	**Resultado**
`x.append(e)`	Agrega a la lista **x** el elemento **e**
`x.insert(i,e)`	Inserta **e** en la posición **i** de la lista **x**
`x.count(e)`	Conteo de instancias de **e** en la lista **x**
`x.remove(e)`	Elimina el primer elemento **e** de la lista **x**
`x.index(e)`	Entrega la posición del primer elemento **e** en **x**
`x.sort()`	Ordena **x** en forma creciente
`x.reverse()`	Invierte la lista **x**
`x.clear()`	Vaciar la lista **x**

Algunas de estas funciones producen un error **ValueError** si no encuentran el elemento, por lo tanto primero debería hacerse una validación.

Operadores y funciones adicionales para listas

`+` Operador de concatenación de listas

`in`
`not in` Operador de inclusión

`del` Elimina elementos de una lista especificando el índice

Ejemplos

```
>>> x=[34,56,75,56,43]
>>> x.append(28)                          Agregar elemento con valor 28
>>> x
[34, 56, 75, 56, 43, 28]

>>> x.insert(2,62)                        Insertar 62 en la posición 2
>>> x
[34, 56, 62, 75, 56, 43, 28]
>>> n=x.count(56)
>>> n
2
>>> x.remove(75)
>>> x
[34, 56, 62, 56, 43, 28]

>>> k=x.index(56)
>>> k
1
>>> x.reverse()
>>> x
[28, 43, 56, 62, 56, 34]
>>> x.clear()
>>> x
[]                                        Lista vacía. Son dos corchetes juntos

>>> x=[3,7,8,2,8,5,4,6]
>>> 9 in x                     Determinar si un elemento está en una lista
False
>>> 8 in x               Si el elemento está en la lista el resultado es True
True
>>> x.index(8)         En este caso se puede determinar su posición en la lista
2
```

Los operadores relacionales también se pueden usar para comparar listas

```
>>> a=[2,3,4]
>>> b=[2,3,1,5]
>>> r=a<b                    La comparación se hace elemento a elemento
>>> r
False
```

Para prevenir errores al usar estas funciones, se puede usar el operador in

Ejemplo. Remover un elemento si está en la lista

```
>>> if 9 in x:
      x.remove(9)
>>> x                          La lista no es modificada y no se produjo un error
[3,7,8,2,8,5,4,6]
```

Si no se hace esta validación y se intenta eliminar un elemento que no está en la lista se genera un error de ejecución: **ValueError**

Ordenamiento de listas: sort

```
>>> x=[34,56,75,56,43]
>>> x.sort()
>>> x
[34, 43, 56, 56, 75]

>>> x=['Química','Algebra','Física','Biología']
>>> x.sort()
>>> x
['Algebra', 'Biología', 'Física', 'Química']

>>> x=[23,'Algebra','Biología',73,45]
>>> x.sort()                   No se pueden ordenar tipos diferentes

TypeError: unorderable types: str() < int()

>>> t = [[7, 3], [3, 6], [9, 8],[5, 4]]          Ordena el primer componente
>>> t.sort()
>>> t
[[3, 6], [5, 4], [7, 3], [9, 8]]

>>> a = [[7, 'Algebra'], [3, 'Física'], [9, 'Química']]
>>> a.sort()
>>> a
[[3, 'Física'], [7, 'Algebra'], [9, 'Química']]
```

El operador + es una alternativa a la función append para concatenar listas

```
>>> x=[5,3,5,2]
>>> x=x+[9,3]
>>> x
[5, 3, 5, 2, 9, 3]
```

```
>>> a=[27,'abc',4.5]
>>> b=[3.2,73,'rt',35]
>>> c=a+b
>>> c
[27, 'abc', 4.5, 3.2, 73, 'rt', 35]
```

Los métodos se aplican igualmente a listas con elementos de diferente tipo

```
>>> x=[23,5.27,'lunes',49]

>>> x.append('martes')
>>> x
[23, 5.27, 'lunes', 49, 'martes']

>>> x.remove('lunes')
>>> x
[23, 5.27, 49, 'martes']

>>> x.index('martes')
3

>>> x=x+['jueves']
>>> x
[23, 5.27, 49, 'martes', 'jueves']
```

Reproducción de listas con *

```
>>> x=[2,5.1,'lunes']
>>> r=2*x                                  Reproduce la lista dos veces
>>> r
[2, 5.1, 'lunes', 2, 5.1, 'lunes']
```

Un operador especial para eliminar elementos de una lista mediante el índice: del

```
>>> x=[3,4,9,7,2,9,6]
>>> del x[3]                               Elimine el elemento en la celda 3
>>> x
[3, 4, 9, 2, 9, 6]
>>> del x[-1]                              Elimine el último elemento
>>> x
[3, 4, 9, 2, 9]

>>> x=[23, 5.27, 'lunes', 49, 'martes']
>>> del x[2]                               Elimine el elemento en la celda 2
>>> x
[23, 5.27, 49, 'martes']
```

Una función para formar pares entre dos listas: zip

```
>>> c=[101,231,725]
>>> m=['Algebra','Física','Química']
>>> p=zip(c,m)
>>> list(p)
[(101, 'Algebra'), (231, 'Física'), (725, 'Química')]
```

Puede ser útil para iterar sobre las dos listas

```
>>> for a,b in p:
      print(a,b)

101 Algebra
231 Física
725 Química
```

Una función para construir listas por mapeo: map

```
>>> def cubo(x):return x**3                Función para calcular un cubo
>>> c=map(cubo, range(5))
>>> s=list(c)
>>> s
[0, 1, 8, 27, 64]
```

Una función para construir listas mediante un filtro: filter

```
>>> def par(x):return x%2==0               Función para detectar número par
>>> x=[3,6,7,8,10,5]
>>> t=filter(par,x)
>>> list(t)
[6, 8, 10]
```

7.7.2 Construcción declarativa de listas numéricas

Las listas numéricas pueden construirse mediante declaraciones en la **ventana interactiva** o **dentro de programas**:

a) Mediante declaraciones implícitas

```
>>> u=range(5)                          Con la función range
>>> x=list(u)
>>> x
[0, 1, 2, 3, 4]

>>> c=[i for i in range(5)]             Crea una lista numérica
>>> c
[0, 1, 2, 3, 4]
```

```
>>> c=[[i] for i in range(5)]          Crea una lista de listas
>>> c
[[0], [1], [2], [3], [4]]
```

```
>>> x=[3,4,7,6,9,8,2]                  Filtrar elementos pares de una lista
>>> p=[t for t in x if t%2==0]
>>> p
[4, 6, 8, 2]
```

Crear una lista de 10 elementos con ceros

```
>>> x=[0 for i in range(10)]
>>> x
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Crear una lista con 8 números aleatorios de una cifra

```
>>> x=[randint(0,9) for i in range(8)]
>>> x
[0, 9, 8, 2, 6, 6, 1, 2]
```

b) Mediante declaraciones explícitas

```
>>> x=[]                               Se inicia como una lista vacía
>>> for i in range(5):                 Crea una lista numérica
        x.append(i)
>>> x
[0, 1, 2, 3, 4]
```

```
>>> x=[]
>>> for i in range(5):                 Crea una lista de listas
        x.append([i])
>>> x
[[0], [1], [2], [3], [4]]
```

Se puede usar el operador **+** para agregar elementos, en lugar de **append**

```
>>> x=[]                               Se inicia como una lista vacía
>>> for i in range(5):                 Crea una lista numérica
        x=x+[i]                        conteniendo enteros
>>> x
[0, 1, 2, 3, 4]
```

```
>>> x=[]
>>> for i in range(5):                 Crea una lista de listas
        x=x+[[i]]                      conteniendo enteros
>>> x
[[0], [1], [2], [3], [4]]
```

El operador **+** o la función **append** también pueden usarse para concatenar listas

```
>>> a=[2,5,7,6]
>>> b=[9,8,4]
>>> c=a+b
>>> c
[2, 5, 7, 6, 9, 8, 4]                Concatenación de elementos
>>> c=a+[b]                          Concatenación de elementos y una lista
>>> c
[2, 5, 7, 6, [9, 8, 4]]
>>> c=[a]+[b]                        Concatenación de listas
>>> c
[[2, 5, 7, 6], [9, 8, 4]]
```

Para crear una lista de listas vacías se puede escribir

```
>>> x=[]                             Se inicia como una lista vacía
>>> for i in range(5):
      x=x+[[]]
>>> x
[[], [], [], [], []]
```

Se pueden crear listas pero sin especificar aún ni los valores ni su tipo

```
>>> x=[]                             Se inicia como una lista vacía
>>> for i in range(5):
      x=x+[None]
>>> x
[None, None, None, None, None]
```

7.7.3 Nombres de listas vinculados

Al asignar el nombre de una lista a otro nombre, ambos se refieren a las mismas celdas

```
>>> x=[20,30,50]
>>> u=x                    Los nombres x, u se refieren a las mismas celdas
>>> x[1]=40
>>> u                      La variable u contiene los mismos valores que x
[20, 40, 50]
```

Si se desea que la lista **u** tenga una copia de la lista **x** con celdas propias, puede crear celdas para **u** y asignar los valores de **x**, o usar la función **copy( )** como en el ejemplo:

```
>>> x=[20,30,50]
>>> u=x.copy()             La variable u tiene una copia de x en otras celdas
>>> x[1]=40
>>> u
[20, 30, 50]
```

7.7.4 Algunas funciones de la librería estándar para listas numéricas

```
>>> x=[3,4,9,7,2,9,6]

>>> len(x)                     Longitud de un vector
7
>>> max(x)                     El mayor valor
9
>>> min(x)                     El menor valor
2
>>> sum(x)                     Suma de los componentes
40
>>> u=[]                       Lista vacía. Su longitud es cero
>>> len(u)
0
```

7.7.5 Algunas funciones de la librería NumPy para listas numéricas

NumPy es una librería de soporte para aplicaciones matemáticas, científicas y de ingeniería. En la sección de matrices se usarán muchas funciones de esta librería para manejo de vectores y matrices. En esta librería, los objetos fundamentales son las listas numéricas, las cuales se denominan arreglos o vectores y matrices. Estos objetos deben definirse con la palabra especial **array( )**

```
>>> from numpy import*
>>> x=array([3,4,9,7,2,9,6])
>>> prod(x)                    Producto de los componentes
81648
>>> argmax(x)                  Índice del mayor valor
2
>>> argmin(x)                  Índice del menor valor
4
>>> mean(x)                    Media aritmética
5.7142857142857144
>>> std(x)                     Desviación estándar
2.6029810226126568
```

Algunas formas de selección lógica de elementos de arreglos

```
>>> a=array([10,14,25,50])
>>> e=a>20                     Expresión lógica para seleccionar elementos
>>> a[e]
array([25, 50])

>>> e=array([0,2,1,1])         Números de celda de elementos elegidos
>>> a[e]
array([10, 25, 14, 14])
```

7.7.6 Salida de listas formateada

Para mostrar listas en pantalla se puede especificar la salida como se describe en el siguiente ejemplo. Se muestra un vector sin formatear y el mismo vector con su salida formateada indicando la cantidad de decimales:

```
>>> x=[1/3,2/5,3/7,4/3]

>>> print(x)
[0.3333333333333333, 0.4, 0.42857142857142855, 1.3333333333333333]

>>> print([float('%6.4f' % x[i]) for i in range(len(x))])
[0.3333, 0.4, 0.4286, 1.3333]
```

7.7.7 Resolución de problemas con listas y vectores

En esta sección se desarrollarán programas y funciones de aplicación con listas principalmente numéricas. Este tipo de estructuras de datos se denominan vectores o arreglos de una dimensión. Los programas y funciones serán escritos como **módulos** en la ventana de edición de Python y probados en la ventana interactiva o desde programas. Los ejemplos desarrollados son una referencia para problemas de aplicación con listas

Algunas aplicaciones también pueden usarse con datos estructurados de otros tipos.

Ejemplo. Escriba un **programa** para sumar los componentes de un vector con valor par

Solución

Variables

- **n**: cantidad de datos
- **x**: contendrá cada dato ingresado
- **v**: vector que contendrá los datos
- **s**: es la suma de los componentes del vector con valor par

Programa

```
# Programa para sumar elementos pares de un vector
n=int(input('Cantidad de elementos: '))
v=[]
for i in range(n):
    x=int(input('Ingrese siguiente dato: '))
    v=v+[x]
s=0
for e in v:
    if e%2==0:
        s=s+e
print('La suma es: ',s)
```

Prueba del programa

```
>>>
Cantidad de elementos: 3
Ingrese siguiente dato: 23
Ingrese siguiente dato: 42
Ingrese siguiente dato: 12
La suma es:  77
```

Ejemplo. Escriba una **función** para sumar los componentes de un vector con valor par

El programa del ejemplo anterior se lo escribe nuevamente como una función.

Solución

Nombre de la función: **vectsumap**
Nombre del módulo: **vectsumap**

Variables

v: vector con los números que ingresan a la función
s: suma de los componentes del vector con valor par

```
def vectsumap(v):
    s=0
    for e in v:
        if e%2==0:
            s=s+e
    return s
```

Prueba de la función desde la ventana interactiva

```
>>> from vectsumap import vectsumap
>>> v=[23,42,12,34,25]
>>> s=vectsumap(v)
>>> s
88
```

NOTA. En Python se pueden resolver muchas aplicaciones usando directamente los operadores y funciones de las librerías disponibles en este lenguaje y que fueron descritos en las secciones anteriores.

El ejemplo anterior se puede resolver en la ventana interactiva con las siguientes instrucciones:

```
>>> x=[23,42,12,34,25]
>>> p=[t for t in x if t%2==0]      genera en forma implícita una lista
>>> p                               con los números pares
[42, 12, 34]
>>> sum(p)                          suma de los números pares
88
```

Ejemplo. Escriba un **programa** para eliminar los elementos repetidos de un vector

Solución

Variables

- **n**: cantidad de datos
- **x**: contendrá cada dato ingresado
- **v**: vector que contendrá los datos
- **u**: vector con elementos diferentes

En la solución, se toma cada elemento del vector original **v** y se lo traslada a otro vector **u** verificando que no haya sido agregado anteriormente.

Programa

```
#Eliminar elementos repetidos de un vector
n=int(input('Cuantos elementos: '))
v=[]
for i in range(n):
    x=int(input('Ingrese el siguiente elemento: '))
    v=v+[x]
print('Lista original: ',v)
u=[]
for e in v:
    if e not in u:
        u=u+[e]
print('Lista depurada: ',u)
```

Prueba del programa

```
>>>
Cuantos elementos: 6
Ingrese siguiente elemento: 7
Ingrese siguiente elemento: 8
Ingrese siguiente elemento: 7
Ingrese siguiente elemento: 9
Ingrese siguiente elemento: 6
Ingrese siguiente elemento: 8

Lista original:  [7, 8, 7, 9, 6, 8]
Lista depurada:  [7, 8, 9, 6]
```

NOTA. En Python se puede resolver esta aplicación convirtiendo el vector a un conjunto y luego convirtiendolo nuevamente a lista:

```
>>> x=[2,3,5,3,6,2,7]
>>> t=set(x)                    El conjunto elimina los elementos repetidos
>>> x=list(t)                   Los elementos de la lista son indexables
>>> x
[2, 3, 5, 6, 7]
```

Ejemplo. Escriba una **función** que reciba una lista y elimine los elementos repetidos.

El programa del ejemplo anterior se lo escribe nuevamente como una función.

Solución

Nombre de la función: **vectrep**
Nombre del módulo: **vectrep**

Variables

v: lista con los datos que entran a la función
u: lista resultante con elementos diferentes

```
#función para eliminar elementos repetidos de una lista
def vectrep(v):
    u=[]
    for e in v:
        if e not in u:
           u=u+[e]
    return u
```

Uso de la función en la ventana interactiva

```
>>> from vectrep import vectrep
>>> x=[2,3,5,3,6,2,7]
>>> y=vectrep(x)
>>> y
[2, 3, 5, 6, 7]
```

Python no requiere que en la función se especifique el tipo de parámetros de entrada, por lo tanto función puede ser usada con datos de diferente tipo como en el siguiente ejemplo:

```
>>> from vectrep import vectrep
>>> x=['abc',37,'abc',47,37]
>>> u=vectrep(x)
>>> u
['abc', 37, 47]
```

Una conclusión derivada de este ejemplo es que en Python es preferible estructurar las soluciones mediante funciones.

Ejemplo. Una **función** para encontrar el mayor valor de un vector y su índice

Solución

La estrategia consiste en iniciar la variable de salida **r**, con primer elemento del vector. Mediante un ciclo se compara cada uno de los otros elementos con el valor asignado a **r**. En cada comparación si el elemento tiene un valor mayor, se actualiza el valor de **r** y también se actualiza la posición o índice de este elemento.

Nombre de la función: **vectmax**
Nombre del módulo: vectmax

Variables
v: vector con los datos que entran a la función
r, p el mayor valor de **v** y su índice. Son los resultados entregados

```
def vectmax(v):
    n=len(v)
    r=v[0]
    p=0
    for i in range(1,n):
        if v[i]>r:
            r=v[i]
            p=i
    return [r,p]
```

Prueba de la función

```
>>> from vectmax import vectmax
>>> x=[11, 5, 18, 16, 14, 11, 13, 3, 18, 16]
>>> [r,p]=vectmax(x)
>>> r
18
>>> p
2
```

NOTA. En Python se puede resolver directamente esta aplicación mediante las funciones definidas en la biblioteca estándar:

```
>>> x=[11, 5, 18, 16, 14, 11, 13, 3, 18, 16]
>>> r=max(x)
>>> r
18
>>> p=x.index(r)
>>> p
2
```

Ejemplo. Escriba en notación abreviada en la ventana interactiva una **función** que entregue el mayor valor y el menor valor de una lista numérica (vector):

```
>>> def fm(x): return [min(x), max(x)]

>>> x=[3,5,2,9,6]
>>> [a,b]=fm(x)
>>> a
2
>>> b
9
```

En los ejemplos anteriores se han desarrollado algunos programas y funciones para manejo de listas. Adicionalmente se han descrito soluciones abreviadas usando las facilidades disponibles en el lenguaje Python.

Se recomienda usar soluciones basada en los operadores y funciones especiales de un lenguaje después que se haya desarrollado la capacidad para programar usando únicamente las instrucciones básicas del lenguaje y unas pocas funciones necesarias.

Esta práctica permite desarrollar el pensamiento algorítmico y la habilidad para programar sus propias soluciones y no se quedará dependiendo de los recursos existentes en un lenguaje computacional particular.

El siguiente es un ejemplo adicional para resaltar lo expresado en el recuadro anterior.

Ejemplo. Escriba una función para ordenar una lista de palabras en forma creciente usando como criterio la longitud de las palabras

La siguiente es una posible solución mediante funciones y operadores disponibles en Python para operar con listas cuyo uso fue descrito en secciones anteriores.

Solución

Función: **sort_len** entrega una lista de palabras ordenadas por su longitud

Variables

x: lista de palabras
v: lista de longitudes de las palabras.
u: lista de pares (longitud, palabra). Usa la función **zip**
h: pares con tipo **list** para ordenar con **sort** por el primer componente
s: lista de las palabras ordenadas (segundo componente de los pares **h**)

```
def sort_len(x):
    v=[]
    for p in x:
        v=v+[len(p)]
    u=zip(v,x)
    h=list(u)
    h.sort()
    s=[]
    for i in range(len(h)):
        s=s+[h[i][1]]
    return s
```

Prueba de la función desde la ventana interactiva

```
>>> from sort_len import sort_len
>>> x=['Algebra','Física','Programación','Lenguajes','Inglés','Química','Biología']
>>> s=sort_len(x)
>>> print(s)
['Física','Inglés','Algebra','Química','Biología','Lenguajes','Programación']
```

Algunos detalles de la solución quedan ocultos al usuario detrás de las funciones Python utilizadas. Estas soluciones deben utilizarse después de desarrollar experiencia en el uso de las instrucciones básicas del lenguaje Python y que las encontraría normalmente en otros lenguajes de programación.

La siguiente es una solución más clara y lógica para el mismo problema anterior. Es el tipo de solución que deben intentar los usuarios hasta que hayan desarrollado su habilidad para programar. Para este ejemplo, la solución propuesta es más simple de entender y además se usan menos variables y menos funciones propias del lenguaje.

Ejemplo. Escriba una función para ordenar una lista de palabras en forma creciente usando como criterio la longitud de las palabras

Solución

Función: **sort_len** Entrega una lista de palabras ordenadas por su longitud

Variables

x: lista de palabras

lpx: lista con las longitudes de las palabras de la lista

Idea propuesta

Se construye una lista con las longitudes de las palabras incluidas en la lista de entrada **x**.

La longitud de cada palabra es comparada con la longitud de las restantes palabras. Cada vez que se encuentra una longitud menor, se intercambian las longitudes y también se intercambian las palabras. de tal manera que el menor valor siempre irá quedando a la izquierda y se mantiene la correspondencia con las palabras.

Al finalizar, la lista de longitudes estará ordenada en forma creciente, y las palabras de la lista también estarán colocadas en este orden.

```
def sort_len(x):
    lpx=[]
    for p in x:
        lpx=lpx+[len(p)]

    for i in range(len(x)):
        for j in range(i+1,len(x)):
            if lpx[j]<lpx[i]:
                lpx[i],lpx[j]=lpx[j],lpx[i]
                x[i],x[j]=x[j],x[i]
    return x
```

Prueba de la función desde la ventana interactiva

```
>>> from sort_len import sort_len
>>> x=['Algebra','Física','Programación','Lenguajes','Inglés','Química','Biología']
>>> s=sort_len(x)
>>> print(s)
['Física','Inglés','Algebra','Química','Biología','Lenguajes','Programación']
```

NOTA. Observe el uso de la asignación que intercambia el contenido de dos variables.

Ejemplo. Una funcion para ordenar una lista en forma creciente

Solución

Nombre de la función: **orden**
Nombre del módulo: **orden**

Variables

x: Lista con los datos que entran a la funcion
La misma variable contiene la lista ordenada resultante

Idea propuesta

Esta solución se basa en el ejemplo anterior. Cada elemento de **x** es comparado con los restantes elementos. Cada vez que se encuentra un elemento con menor valor, se intercambian estos elementos, de tal manera que el menor valor siempre irá quedando a la izquierda. Al finalizar, la lista estará ordenada en forma creciente. Debido a que la representación interna de los caracteres tiene una representación mediante códigos ordenados, esta función puede usarse también con caracteres o cadenas de caracteres

```
def orden(x):
    for i in range(len(x)):
        for j in range(i+1,len(x)):
            if x[j]<x[i]:
                x[i],x[j]=x[j],x[i]
    return x
```

Prueba de la función en la ventana interactiva

```
>>> from orden import orden
>>> x=[2,5,6,9,3,4,5]
>>> v=orden(x)
>>> v
[2, 3, 4, 5, 5, 6, 9]
>>> x=['r','t','u','a','c']
>>> v=orden(x)
>>> v
['a', 'c', 'r', 't', 'u']
>>> x=['arte','algebra','banco','artesano','cabos','base']
>>> v=orden(x)
>>> v
['algebra', 'arte', 'artesano', 'banco', 'base', 'cabos']
```

Pregunta para investigar: Si la lista tiene longitud **n**, determine la cantidad total de ciclos que se realizan en la función, para ordenarla.

Python tiene una función para ordenar listas:

```
>>> v=[2,5,6,9,3,4,5]
>>> v.sort()
>>> v
[2, 3, 4, 5, 5, 6, 9]
```

Ejemplo. Una funcion alterna para ordenar una lista en forma creciente

Solución

Nombre de la función: **ordena**

Nombre del módulo: **ordena**

Variables

x: Lista con los datos que entran a la funcion
La misma variable contiene la lista ordenada resultante

p: Posición del elemento con el menor valor en cada sublista de izquierda a derecha

Idea propuesta

Existen muchos algoritmos para ordenar una lista. En la siguiente propuesta, la instrumentación se fundamenta en la siguiente idea: Cada elemento **x[i]** es intercambiado con el elemento que tiene el menor valor buscando entre los elementos **x[j], j=i, i+1, i+2, ..., n-1**. Al completar este proceso con todos los elementos **x[i]** de izquierda a derecha, la lista quedará ordenada en forma creciente.

```
def ordena(x):
    n=len(x)
    for i in range(n-1):
        t=x[i]
        p=i
        for j in range(i+1,n):
            if x[j]<t:
                t=x[j]
                p=j
        x[i],x[p]=x[p],x[i]
    return x
```

Prueba de la función en la ventana interactiva

```
>>> from ordena import ordena
>>> x=[2,5,6,9,3,4,5]
>>> v=ordena(x)
>>> v
[2, 3, 4, 5, 5, 6, 9]
```

Ejemplo. Escriba una función para generar un vector con números aleatorios enteros en un rango especificado

Solución

Nombre de la función: **vectrandint**
Nombre del módulo: **vectrandint**
Parámetros: **n, a, b** longitud del vector y extremos del rango
Resultado: **v** vector con los números aleatorios

La función usa la función randint del módulo random

```
from random import*
def vectrandint(n,a,b):
    v=[]
    for i in range(n):
        v=v+[randint(a,b)]
    return v
```

Prueba de la función: Almacene en una lista 10 resultados de lanzamientos de un dado

```
>>> from vectrandint import vectrandint
>>> x=vectrandint(10,1,6)
>>> x
[2, 1, 1, 3, 2, 4, 2, 6, 3, 6]
```

El vector resultante puede incluir elementos repetidos

Ejemplo. Desde la ventana interactiva llamar a la función **vectrandint** para obtener un vector de 10 números aleatorios enteros entre 1 y 20. Luego elimine los elementos repetidos.

```
>>> from vectrandint import vectrandint
>>> x=vectrandint(10,1,20)
>>> x
[7, 3, 10, 6, 15, 18, 1, 16, 1, 6]
```

El vector puede incluir repeticiones

Para eliminar los resultados repetidos se puede proceder de la siguiente manera

```
>>> s=set(x)
>>> s
{1, 3, 6, 7, 10, 15, 16, 18}
>>> s[2]
TypeError: 'set' object does not support indexing
```

Al convertir a conjunto se eliminan repeticiones

El resultado es un conjunto pero **no indexable**

Se puede convertir en una lista, que si es indexable

```
>>> u=list(s)
>>> u
[1, 3, 6, 7, 10, 15, 16, 18]
>>> u[2]
6
```

Se puede transformar nuevamente a lista

La lista es una estructura de datos **indexable**

Si se desea desordenar la lista se puede usar una función de la librería **random**

```
>>> from random import*
>>> shuffle(u)
>>> u
[3, 10, 6, 18, 16, 7, 15, 1]
```

La función **shuffle** desordena la lista

Ejemplo. Crear una función para generar una muestra aleatoria de tamaño **m** con enteros tomada de una población de tamaño **n**. Una muestra aleatoria es una lista de números aleatorios pero sin que contenga resultados repetidos.

Solución

Nombre de la función: **muestra**
Nombre del módulo: **muestra**
Parámetros: **n, m**
Resultado: **v** vector con los componentes de la muestra aleatoria

Idea propuesta: Mientras el vector de resultados no esté lleno, agregar cada número aleatorio al vector pero si no está incluido.

```
#Generar una muestra aleatoria
from random import*
def muestra(n,m):
    v=[]
    while len(v)<m:
        x=randint(1,n)
        if x not in v:
            v=v+[x]
    return v
```

Prueba de la función desde la ventana interactiva

```
>>> from muestra import muestra
>>> v=muestra(20,10)
>>> v
[8, 11, 19, 1, 7, 15, 17, 3, 5, 4]
```

El resultado es una lista con números elegidos **aleatoriamente** y **sin repeticiones.**

Cada vez que se pruebe este programa se obtendrá una nueva muestra aleatoria.

Con la función **sample** de la **random** se pueden obtener directamente **muestras aleatorias**:

```
>>> from random import*
>>> p=list(range(20))
>>> v=sample(p,5)
>>> v
[6, 0, 9, 11, 19]
```

genera la lista: **[0, 1, 2, . . ., 19]**
muestra aleatoria de tamaño **5**
tomada de la lista **p**

Ejemplo. La secuencia de Ulam es una sucesión de números naturales generados a partir de un dato inicial con la siguiente definición.

$$x = \begin{cases} x/2, & x \text{ par} \\ 3x+1, & x \text{ impar} \end{cases}, \quad x \text{ es un número natural}$$

Aplicando recurrentemente esta deficinión, siempre llegará finalmente al número 1

Caso: Si el valor inicial es **x=12**, los valores sucesivos serán: **6, 3, 10, 5, 16, 8, 4, 2, 1**

Escriba un programa que lea un número y muestre la secuencia pero en **orden inverso**.

Solucíon

Variables

n: valor inicial para la secuencia. **n** es modificado con la regla indicada
u: vector con los números de la secuencia de Ulam invertida

Cada nuevo elemento de la secuencia será agregado a la izquierda en el vector **u**. El vector resultante contendrá los elementos ubicados en el orden inverso, es decir la secuencia de los números de Ulam comenzará desde 1 hasta llegar al valor inicial.

Programa

```
#secuencia de Ulam invertida
n=int(input('Ingrese un entero positivo: '))
u=[]
while n>1:
    if n%2==0:
        n=n//2
    else:
        n=3*n+1
    u=[n]+u
print(u)
```

Prueba del programa

```
>>>
Ingrese un entero positivo: 12
[1, 2, 4, 8, 16, 5, 10, 3, 6]
```

Ejemplo. Obtenga el conteo de frecuencias para los siguientes 40 datos de una muestra correspondientes al tiempo que se utilizó para atender a las personas en una estación:

3.1	4.9	2.8	3.6	4.5	3.5	2.8	4.1
2.9	2.1	3.7	4.1	2.7	4.2	3.5	3.7
3.8	2.2	4.4	2.9	5.1	1.8	2.5	6.2
2.5	3.6	5.6	4.8	3.6	6.1	5.1	3.9
4.3	5.7	4.7	4.6	5.1	4.9	4.2	3.1

Los datos serán distribuidos en 6 intervalos o clases con la siguiente definición:

```
[1, 2), [2, 3), [3, 4), [4, 5), [5, 6), [6, 7)
```

Solución

Variables

x: lista con los datos observados
c: lista de listas con los intervalos de las clases
f: conteo de datos en cada clase

Se definirá una función con el nombre **frecuencia** para realizar el conteo de datos en cada clase. Las clases son una lista cuyos componentes son listas con los extremos de clase:

```
c=[[1,2],[2,3],[3,4],[4,5],[5,6],[6,7]]
```

En un programa escrito junto a la función se definen los datos y se imprimen los resultados. Si las pruebas se hicieran desde la ventana interactiva, habría que repetir cada vez el ingreso de los datos

```
from numpy import*
def frecuencia(c,x):
    f=zeros(len(c))
    for e in x:
        for j in range(len(c)):
            if e>=c[j][0] and e<c[j][1]:
                f[j]=f[j]+1
    return f

c=[[1,2],[2,3],[3,4],[4,5],[5,6],[6,7]]
x=[3.1, 4.9, 2.8, 3.6, 4.5, 3.5, 2.8, 4.1,
   2.9, 2.1, 3.7, 4.1, 2.7, 4.2, 3.5, 3.7,
   3.8, 2.2, 4.4, 2.9, 5.1, 1.8, 2.5, 6.2,
   2.5, 3.6, 5.6, 4.8, 3.6, 6.1, 5.1, 3.9,
   4.3, 5.7, 4.7, 4.6, 5.1, 4.9, 4.2, 3.1]

f=frecuencia(c,x)
print('Conteo de frecuencia por clase')
print(f)
```

Prueba del programa

```
>>>
Conteo de frecuencia por clase
[ 1.   9.  11.  12.   5.   2.]
```

Ejemplo. Realice el conteo de frecuencia y grafique mediante barras 200 datos con distribución Normal o Gaussiana con media **5** y desviación estándar **2**. Las clases se definen en la prueba y los datos son obtenidos con la función **gauss** de la librería random.

```
from numpy import*
def frecuencia(c,x):
    f=zeros(len(c))
    for e in x:
        for j in range(len(c)):
            if e>=c[j][0] and e<c[j][1]:
                f[j]=f[j]+1
    return f

def barras(f):
    for e in f:
        barra=''
        for j in range(1,int(e)+1):
            barra=barra+'*'
        print('%4d '%(e),barra)

from random import*
c=[[-2,-1],[-1,0],[0,1],[1,2],[2,3],[3,4],
   [4,5],[5,6],[6,7],[7,8],[8,9],[9,10],[10,11]]
x=[]
for i in range(200):
    x=x+[gauss(5,2)]
f=frecuencia(c,x)
print('Conteo de frecuencia por clase')
print(f)
print('Diagrama de barras')
barras(f)
```

Prueba del programa

```
>>>
Conteo de frecuencia por clase
[  0.   2.   6.   8.  18.  29.  39.  44.  30.  16.   5.   1.   1.]
Diagrama de barras
    0
    2  **
    6  ******
    8  ********
   18  ******************
   29  *****************************
   39  ***************************************
   44  ********************************************
   30  ******************************
   16  ****************
    5  *****
    1  *
    1  *
```

7.7.8 Ejercicios con listas y vectores

1. Una persona tiene una lista con los precios de n artículos y dispone de una cierta cantidad de dinero. Escriba un programa para leer estos datos y almacenarlos en un vector:

a) Muestre los numeros de los artículos que puede comprar
b) Para cada artículo cuyo precio es menor que la cantidad de dinero disponible, determine cuantas unidades puede comprar

2. Lea una lista de los pesos de las **n** cajas en un contenedor. Determine cuantas cajas tienen el peso mayor al peso promedio del grupo.

3. Lea una lista de los pesos de los **n** objetos en una bodega. Determine cual es el rango de los pesos de estos objetos.

4. Una bodega contiene **n** paquetes numerados en forma natural. Para una inspección se debe tomar una muestra aleatoria del **10%** de los paquetes. Escriba un programa para elegir la muestra. La muestra no debe contener elementos repetidos.

5. Para la inspección de los **m** paquetes de una bodega se han elegido a **m** inspectores. Realice aleatoriamente la asignación de tal manera que cada inspector se le asigne la revisión de un solo paquete.

6. Para la inspección de los **m** contenedores de una bodega se han elegido a **m/2** inspectores. Realice aleatoriamente la asignación de tal manera que cada inspector se le asigne la revisión de dos contenedores. No se puede asignar un contenedor más de una vez.

7. Escriba un programa para leer un vector con números enteros de una cifra. Luego genere un número aleatorio de una cifra. Entonces, elimine del vector todos los números cuyo valor sea menor al número aleatorio. Muestre el vector resultante.

8. Lea la lista de los números de identificación de los estudiantes que están inscritos en la material A, y otra lista con los estudiantes que están inscritos en la materia B.

a) Encuentre cuantos estudiantes están inscritos en la materia A y también en la materia B
b) Encuentre los estudiantes que están inscritos en la materia A pero no en la materia B

9. Se tiene la lista de los códigos de las cajas en la bodega A y una lista de los códigos de las cajas en la bodega B. Además se tiene una lista con los códigos de las cajas que deben ser inspeccionadas. Determine cuantas cajas serán inspeccionadas en cada bodega.

10. Almacene en un vector **X** las abscisas y en un vector **Y** las ordenadas de un conjunto de puntos en un plano. Ambos vectores son ingresados como datos. Suponga que estos puntos representan los vértices de un polígono. Determine el perímetro del polígono.

Use la fórmula para calcular la distancia entre cada par de puntos.

$$d = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

11. Almacene en un vector **X** las abscisas y en un vector **Y** las ordenadas de un conjunto de puntos en un plano. Ambos vectores son ingresados como datos. Determine cual es el punto más alejado del origen. Use la fórmula de la distancia. $\mathbf{d} = \sqrt{\mathbf{x}_i^2 + \mathbf{y}_i^2}$

12. El cálculo del área de un polígono arbitrario, dadas las coordenadas de sus vértices **$(x_1, y_1)(x_2, y_2), ..., (x_n, y_n)$** se lo puede hacer con la siguiente fórmula:

$$\text{Area}= [(x_1+x_2)(y_1-y_2) + (x_2+x_3)(y_2-y_3)+ ... +(x_{n-1}+x_{n-2})(y_{n-1}-y_{n-2}) + (x_n+x_1)(y_n-y_1)]/2$$

Escriba un programa para leer en vectores los valores de las abscisas y ordenadas de los vértices. Calcule y muestre el valor del área con la fórmula anterior.

13. Para simular los saltos de **n** ranas en una pista de longitud **m** metros se usará un vector de **n** elementos que inicialmente contiene ceros. Use un ciclo para agregar a cada rana un número aleatorio que puede ser **0**, **1** o **2** metros. Repita este ciclo hasta que alguna rana llegue al final de la pista. Muestre en cual turno alguna rana llegó al final de la pista.

14. En un proceso electoral se tienen anotados los **n** votos para aprobar una moción. Cada voto tiene el número de identificación del elector (números enteros del **1** al **n**) y un número que representa su decisión: 1 si es a favor, 0 si es en contra, cualquier otro número es nulo. Escriba un programa que lea los **n** datos conteniendo el número del elector (no suponga que están ordenados) y su voto. Coloque los números de identificación en tres listas: votantes a favor, votantes en contra y votantes nulos. Finalmente busque y muestre si hay números de identificación de electores que están en más de una lista.

15.- Se tiene como dato la cantidad de boletos disponibles para un concierto. Escriba un programa para organizar la venta en línea. Cada fan debe presentar su cédula de identidad. El programa debe leer y agregar a un vector el número de cédula. En caso de que el número de cédula ya esté en el vector, muestre un mensaje rechazando la venta de boletos. Si la venta se realiza, lea la cantidad de boletos que se compra (no debe ser mayor a 4) y reste de la cantidad disponible. Cuando esta cantidad llegue a cero, muestre un mensaje y finalice el programa.

16.- Escriba un programa con un menú para registrar estudiantes en uno de los dos paralelos de una materia mediante las opciones indicadas a continuación. Cada paralelo debe ser representado mediante un vector.

1) Registrar
   Lea el numero del paralelo elegido (1 o 2), luego lea el código del estudiante y agréguelo al vector correspondiente
2) Consultar
   Lea el código del estudiante, búsquelo en los vectores y muestre el paralelo en el que está registrado
3) Cambiar
   Lea el código del estudiante. Si está registrado elimínelo del vector y agréguelo al otro vector
4) Salir

17. En una escuela de fútbol se inscriben **n** jugadores identificados con su número en la lista de asistencia y un código que identifica su habilidad **1**: portero, **2**: defensa, **3**: mediocampista, **4**: delantero. Este dato debe ser ingresado y validado.

Escriba un algoritmo para ayudar al entrenador a formar dos equipos de **11** jugadores elegidos aleatoriamente. Cada equipo debe contener **1** portero, **4** defensas, **2** mediocampistas y **4** delanteros. Cada jugador no puede pertenecer a más de un equipo. Suponga que en la lista hay suficientes jugadores para elegir.

18. Escriba un programa para organizar los 32 equipos participantes en un campeonato en 8 grupos de 4 equipos. La elección de los equipos debe ser aleatoria pero cada grupo debe tener un equipo fijo que debe ser un dato inicial. Los equipos no pueden asignarse más de una vez en los grupos.

19. Escriba una función **divisores(n)** que entregue un vector conteniendo todos los divisores enteros positivos que tiene un número entero dado n. Escriba un programa de prueba que use la función escrita para encontrar para cada número entero del 20 al 30, sus divisores enteros

20. Escriba una función **primos(v)** que reciba un vector **v** y entregue otro vector conteniendo los elementos que son números primos.

21. La operación **xor** en el sistema binario produce el siguiente resultado

m	k	m xor k
0	0	0
0	1	1
1	0	1
1	1	0

Esta operación se usa para encriptar mensajes en binario en los cuales **m** representa el mensaje, **k** la clave para encriptar el mensaje, y **e** el mensaje encriptado.

Ejemplo. Mensaje que se envía: m = 11011001
Clave: k = 01100011
Mensaje encriptado: e = 10111010

El receptor del mensaje encriptado, aplicando la misma clave puede conocer el mensaje:

Ejemplo. Mensaje encriptado: e = 10111010
Clave: k = 01100011
Mensaje recibido: m = 11011001

Escriba una función que reciba un vector conteniendo números en el sistema binario y entregue otro vector conteniendo los números binarios con la operación xor. Esta función se usará para encriptar un mensaje y para conocer el menaje enviado. La función debe validar que los vectores contengan números binarios, caso contrario, el resultado es un vector nulo.

22. Escriba un a función **rango(v,a,b)** que reciba un vector **v**, y determine cuantos elementos están en el rango **[a, b]**

23. Escriba una función que reciba un vector y entregue como resultado otro vector conteniendo los mismos elementos del vector ingresado pero con las elementos ubicados aleatoriamente en otro orden

Ejm. Entra **[3, 7, 6, 2, 9, 8]**, sale **[6, 8, 3, 2, 7, 9]**

Escriba un programa que llene un vector de **n** números aleatorios de una cifra. El programa debe enviar el vector a la función creada y recibir otro vector. El programa debe determinar cuantos números coinciden en la misma posición en ambos vectores

24. Las listas A y B tienen n componentes enteros, mientras que la lista C tiene n pares ordenados ($x \in A$, $y \in B$).

Determine si la correspondencia definida por C es inyectiva y sobreyectiva

25. Escriba una función **sumacum(x)** que entregue un vector con la suma acumulada de un vector

Ejemplo.
```
>>> x=[2,5,8,20,50]
>>> s=sumacum(x)
>>> s
[2,7,15,35,85]
```

26. Un grupo de n personas debe elegir a su representante. Será elegida la persona que tenga más de la mitad de los votos, caso contrario se debe repetir la votación. Cada persona es identificada con un número entero entre 1 y n, y cualquier persona pueden ser elegida. Luego de realizar la votación, se debe realizar el conteo de los votos y validar si es necesario realizar una nueva votación.

7.8 Operaciones con cadenas de caracteres

En esta sección se establecen los instrumentos para desarrollar aplicaciones con cadenas de caracteres o strings.

7.8.1 Cadenas de caracteres constantes

La clase **string** contiene cadenas de caracteres constantes que pueden ser de utilidad como referencia en algunas aplicaciones.

Para usar estas constantes debe importar la clase **string**.

La siguiente es la lista de constantes disponibles:

ascii_letters = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
ascii_lowercase = 'abcdefghijklmnopqrstuvwxyz'
ascii_uppercase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
digits = '0123456789'
hexdigits = '0123456789abcdefABCDEF'
octdigits = '01234567'
printable = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTU...
punctuation = '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
whitespace = ' \t\n\r\x0b\x0c'

Ejemplo.

```
>>> from string import*
>>> x=digits
>>> x
'0123456789'
```

7.8.2 Métodos, operadores y funciones para cadenas de caracteres

En Python, las listas son inmutables, esto significa que no pueden modificarse sus componente mediante una asignación directa como en las listas, pero existen funciones que facilitan el manejo de este tipo de datos.

Para manejo de cadenas de caracteres Python tiene una clase denominada **str**. Esta clase es residente y no necesita importarse para usarla.

Al crear una variable u objeto de tipo cadena de caracteres se tiene acceso a las funciones o métodos definidos en la clase **str** y se los usa con la siguiente notación: **objeto.método**

La clase **str** contiene una gran cantidad de funciones o métodos para operar con cadenas de caracteres. Algunos métodos disponibles en esta clase:

Sea **s**: objeto de tipo cadena de caracteres

Método	Resultado
`s.capitalize()`	Cadena con la primera letra mayúscula y las demás minúsculas
`s.center(n,c)`	Cadena centrada en n espacios. El relleno es un carácter opcional c
`s.count(t,a,b)`	Cantidad de ocurrencias de la subcadena t en la cadena s. Opcionalmente se puede indicar el rango de búsqueda: a,b
`s.find(t,a,b)`	Menor índice de la subcadena t en s. Se puede indicar el rango de búsqueda: a, b, si no lo encuentra retorna -1
`s.isalpha()`	Retorna True si todos los caracteres son alfabéticos
`s.isdigit()`	Retorna True si todos los caracteres son dígitos
`s.islower()`	Retorna True si todos los caracteres son minúsculas
`s.isupper()`	Retorna True si todos los caracteres son mayúsculas
`s.ljust(n,c)`	Cadena justificada a la izquierda en n espacios. El relleno es un carácter opcional c
`s.lower()`	Cadena convertida a minúsculas
`s.lstrip()`	Cadena con espacios a la izquierda eliminados
`s.replace(t,r,n)`	Cadena con la subcadena t reemplazada por la subcadena r. opcionalmente se puede indicar la cantidad de reemplazos n
`s.rfind(t,a,b)`	Mayor índice de la subcadena t en s. Opcionalmente se puede indicar el rango de búsqueda: a,b. Si no lo encuentra retorna -1
`s.rjust(n,c)`	Cadena justificada a la derecha en n espacios. El relleno es un carácter opcional c
`s.split(c)`	Entrega una lista cuyos componentes son las palabras de s separadas con un carácter c
`s.rstrip()`	Cadena con los espacios a la derecha eliminados
`s.strip()`	Cadena con los espacios a la derecha e izquierda eliminados
`s.upper()`	Cadena convertida a mayúsculas

`len(s)`	Función para determinar la cantidad de caracteres en la cadena **s**
`+`	Operador de concatenación de cadenas
`in`, `not in`	Operador de inclusión

Ejemplos.

```
>>> x=''
```
Un carácter nulo son dos comillas juntas

```
>>> s='esta es una prueba'
>>> s.count('e')
3
```
Conteo de coincidencias

```
>>> s.find('ta')
2
```
Búsqueda del índice de una coincidencia

```
>>> s.replace('e','E')
'Esta Es una pruEba'
```
Reemplazo de subcadenas en una cadena

```
>>> u=s.upper()
>>> u
'ESTA ES UNA PRUEBA'
```

```
>>> len(s)
18
```
Longitud de la cadena

```
>>> 'TA' in u
True
```
Operador de pertenencia

```
>>> u=u+' FINAL'
>>> u
'ESTA ES UNA PRUEBA FINAL'
```
Concatenación de cadenas

```
>>> u[2:4]
'TA'
```
Las cadenas se pueden indexar

```
>>> u[3]= 'L'
```
Error: no se puede modificar una cadena por asignación

TypeError: 'str' object does not support item assignment

Borrar elementos de una cadena

```
>>> m='programa'
>>> s=m.replace('a','')              Eliminar todas las ocurrencias de la letra 'a'
>>> s
'progrm'
>>> s=m.replace('a','',1)            Eliminar la primera ocurrencia de la letra 'a'
>>> s
'progrma'
>>> s=m[:4]+m[5:]                    Eliminar el carácter en la celda 4
>>> s
'progama'

>>> s=m[1:]                          Eliminar el primer carácter
>>> s
'rograma'
>>> s=m[:-1]                         Eliminar el último carácter
>>> s
'program'

>>> m='Esta es una prueba'
>>> s=m.replace(' ','')              Eliminar los espacios en blanco de la cadena
>>> s                                Los espacios ' ' son reemplazadoc
'Estaesunaprueba'                    por un carácter nulo ''
```

Los operadores relacionales se pueden usar para comparar cadenas de caracteres

```
>>> x='abc'
>>> y='abde'
>>> r=x<y                            La comparación es carácter con carácter
>>> r                                en el orden alfabético
True
```

Ejemplos adicionales con las funciones de la librería estándar para cadenas

```
>>> s='     abc'
>>> t=s.lstrip()                     Elimina espacios a la izquierda
>>> t
'abc'

>>> s='abcd'
>>> t=s.capitalize()                 Inicial mayúscula
>>> t
'Abcd'
```

```
>>> s='     abc       '
>>> t=s.strip()                        Elimina espacios en ambos lados
>>> t
'abc'

>>> s='abc'
>>> t=s.rjust(10)                      Ajusta la cadena a la derecha en 10 columnas
>>> t
'       abc'

>>> s='abc,rs,tu,xyz'
>>> t=s.split(',')                     Convierte una cadena en una lista
>>> t
['abc', 'rs', 'tu', 'xyz']

>>> s='programa'
>>> '-'.join(s)                        Insertar separadores de caracteres
'p-r-o-g-r-a-m-a'

>>> s='programa'
>>> c=ord(s[0])                        Representación ASCII de un carácter
>>> c
112

>>> t=chr(112)                         Representación como carácter de un código
>>> t
'p'

>>> >>> c=[ord(x) for x in s]          Lista implícita de conversión
>>> c
[112, 114, 111, 103, 114, 97, 109, 97]

>>> s='programa'                       Construcción de una lista desde una cadena
>>> list(s)
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'a']
```

7.8.3 Resolución de problemas con cadenas de caracteres

Ejemplo. Escriba un programa que lea una cadena y la muestre con los caracteres ubicados en orden inverso

Solución

Los caracteres de la cadena ingresada son colocados en otra cadena, inicialmente vacía, agregándolos de derecha a izquierda

```
#Invertir una frase
s=input('Ingrese una frase: ')
t=''
for c in s:
    t=c+t
print(t)
```

Prueba del programa

```
>>>
Ingrese una frase: Esta es una prueba
abeurp anu se atsE
```

Ejemplo. Reescriba el programa de la solución del ejemplo anterior como una función

Solución

Nombre de la función: **strinvertir**
Nombre del módulo: **strinvertir**
Parámetro: **s** cadena
Resultado: **t** cadena invertida

```
def strinvertir(s):
    t=''
    for c in s:
        t=c+t
    return t
```

Prueba de la función en la ventana interactiva

```
>>> from strinvertir import strinvertir
>>> x='Esta es una prueba'
>>> y=strinvertir(x)
>>> y
'abeurp anu se atsE'
```

Ejemplo. En la ventana interactiva resuelva el siguiente problema: Dada una frase, determine si es un **palíndromo**.

Un **palíndromo** es una frase que se puede leer igual de izquierda a derecha o de derecha a izquierda. Los espacios en blanco ni las tildes cuentan para esta definición, tampoco importa si son mayúsculas o minúculas

Solución

```
>>> s='Anita lava la tina'            Frase original
>>> s=s.upper()                       Cambiar a mayúsculas
>>> s
'ANITA LAVA LA TINA'

>>> s=s.replace(' ','')               Eliminar espacios
>>> s
'ANITALAVALATINA'

>>> from strinvertir import strinvertir
>>> t=strinvertir(s)                  Invertir la frase
>>> t
'ANITALAVALATINA'

>>> r=s==t                            Comparar ambas frases
>>> r
True                                  Verifica que es un palíndromo
```

Ejemplo. Leer una frase y mostrarla encriptada intercambiando parejas consecutivas de caracteres

```
def strcripto(s):
    r=''
    n=len(s)
    for i in range(0,n-1,2):
        a=s[i]
        b=s[i+1]
        r=r+b+a
    if n%2!=0:
        r=r+s[n-1]
    return r
```

Prueba de la función

```
>>> from strcripto import strcripto
>>> s='programas'
>>> r=strcripto(s)
>>> r
'rpgoarams'                           Mensaje encriptado
```

```
>>> t=strcripto(r)
>>> t
'programas'
```

La misma función restaura el mensaje original ingresando el mensaje encriptado

Ejemplo. Lea nombres que pueden tener diferente longitud y forme una lista con ellos. Muestre por cada nombre, la cantidad de veces que contiene la letra 'a'

```
#Manejo de una lista de cadenas de caracteres
n=int(input('Cantidad de nombre: '))
s=[]
for i in range(n):
    x=input('Ingrese nombre: ')
    s=s+[x]
for x in s:
    c=x.count('a')
    print(x,c)
```

```
>>>
Cantidad de nombre: 3
Ingrese nombre: doris
Ingrese nombre: anita
Ingrese nombre: carmen
doris 0
anita 2
carmen 1
```

Ejemplo. Lea una lista de nombres. Elimine los nombres con el carácter **'a'**

```
#Manejo de una lista de cadenas de caracteres
n=int(input('Cantidad de nombre: '))
s=[]
for i in range(n):
    x=input('Ingrese nombre: ')
    s=s+[x]
t=[]
for e in s:
    if 'a' not in e:
        t=t+[e]
print(t)
```

```
>>>
Cantidad de nombre: 4
Ingrese nombre: doris
Ingrese nombre: anita
Ingrese nombre: carmen
Ingrese nombre: roxy
['doris', 'roxy']
```

Ejemplo. Escriba una función de utilidad que reciba una cadena conteniendo números separados por comas y los convierta a un vector o lista numérica.

Solución

Nombre de la función: **convnum**
Nombre del módulo: **convnum**
Parámetro: **c** cadena
Resultado: **x** vector

Python no facilita el ingreso directo de una lista numérica pero provee suficientes definiciones para instrumentar una función que permita resolver este problema.

El dato ingresado es una **cadena de caracteres** con números separados por comas. La función **split** la transforma a una lista de cadenas de caracteres, en la cual cada elemento es una subcadena conteniendo cada número.

Mediante un ciclo se convierte cada elemento a tipo entero con el tipo **int** (puede ser tipo real escribiendo **float** en lugar de **int**) y se crea la lista numérica o vector para entregar.

Función

```
#Convertir lista de números de literal a numérico
def convnum(c):
    numc=c.split(',')
    x=[]
    for e in numc:
        x=x+[int(e)]
    return x
```

Programa de prueba de la función convnum

```
#Programa de prueba de convnum
from convnum import convnum
c=input('Ingrese la lista de números: ')
x=convnum(c)
s=0
for n in x:
    s=s+n
print('Suma del vector: ',s)
```

Prueba del programa en la ventana interactiva

```
>>>
Ingrese la lista de números: 23,456,7,4375
Suma del vector:  4861
>>>
```

Nota: los datos son una línea de texto la cual contiene números separados por comas, pero son recibidos como una cadena de caracteres.

Ejemplo. Escriba una función de utilidad que reciba un vector o lista numérica y entregue una cadena de caracteres cuyos elementos son subcadenas conteniendo los números separados por comas

Solución

Nombre de la función: **convstr**
Nombre del módulo: **convstr**
Parámetro: **v** vector
Resultado: **s** cadena

Esta función es inversa a la función **convnum** desarrollada en la sección anterior.

Mediante un ciclo se convierte cada elemento numérico a su representación como cadena con el tipo **str**. Se insertan comas intermedias para separar las subcadenas.

Función

```
#Conversión de un vector a una cadena
def convstr(v):
    s=''
    for x in v:
        s=s+str(x)+','
    return s
```

Prueba de las funciones convnum y convstr en la ventana interactiva

```
>>> from convnum import convnum
>>> c='23,37,56,48'
>>> v=convnum(c)
>>> v
[23, 37, 56, 48]

>>> from convstr import convstr
>>> v=[23, 37, 56, 48]
>>> x=convstr(v)
>>> x
'23,37,56,48,'
```

Estas funciones se pueden usar opcionalmente para almacenar y recuperar listas numéricas que se almacenan como una linea de texto en un archivo.

7.8.4 Ejercicios con cadenas de caracteres

1. Escriba un programa que realice lo siguiente

a) Lea una frase.
b) Cuente y muestre cuantas vocales tiene la frase
c) Lea una palabra, cuente y muestre cuantas veces la frase contiene a la palabra
d) Elimine todas las vocales que contiene la frase. Muestre la frase final

2. Escriba un programa que lea una dirección de correo electrónico con formato **usuario@dominio.tipo** y muestre separadamente cual es el usuario y el tipo

3. Escriba un programa que lea una frase y enmascárela sustituyendo las vocales con símbolos: 'a' sustituya con '*', 'e' con '-', 'i' con '?', 'o' con '&', 'u' con '#'
Escriba otro programa que haga la sustitución inversa y restaure la frase original.

4. Una cadena ADN es una linea de texto conteniendo una lista de los caracteres A, C, G, T en cualquier secuencia. Ejemplo. CCGAATCGTA

Se considera que cada par de caracteres consecutivos está ordenado si el carácter a la izquierda es alfabéticamente menor o igual que el carácter a la derecha.

Escriba un programa que reciba una cadena ADN y muestre cuantos pares de la cadena están ordenados. Verifique que la cadena tenga caracteres válidos, caso contrario, muestre un mensaje.

5. Rescriba un programa que reciba una palabra y desordene las letras en forma aleatoria.

Ejemplo: Recibe 'martes', entrega 'remsta' (un ejemplo)

Sugerencia: Para cada letra, seleccione aleatoriamente otra letra de la palabra con la que intercambiarán posiciones, pero elimínela de la palabra para que no sea elegida otra vez.

6. Escriba un programa para jugar el juego del ahorcado entre una persona y el computador. Primero almacene una lista de palabras en un vector. Luego el computador selecciona una palabra aleatoriamente pero no la muestra. La persona trata en intentos sucesivos adivinar la palabra ingresando una letra en cada intento. El computado muestra las letras que coinciden con la palabra seleccionada, pero en cada fallo, muestra un mensaje que acerca a la paersona a ser ahorcado.

7. Un paso importante en la decodificación de mensajes encriptados es encontrar algunas letras utilizadas. Para ello se debe determinar la frecuencia de los símbolos usados y asociarlos a las letras del alfabeto. Por ejemplo, en el español la letra de mayor frecuencia es la letra e. Escriba un programa que lea un mensaje y determine la frecuencia de cada símbolo utilizado.

8. Escriba una función **codificar(x,k)** que reciba una cadena **x** y una constante **k** y entregue otra cadena con los caracteres desplazados **k** posiciones en el alfabeto. **k** puede ser positivo para codificar o negativo para decodificar. Al exceder el final o el inicio del albabeto, el desplazamiento debe continuar en el otro extremo.

9. Escriba una **función recursiva** que permita invertir una cadena de caracteres.

10. En una expresión aritmética en notación INFIX se escriben los **operandos** (números) separados por un **operador** aritmético conocido (+, –, *, /).
En una expresión aritmética en notación POSTFIX, primero se escriben los operandos y luego el operador, como se muestra en los ejemplos.

INFIX	POSTFIX
2 + 3	2 3 +
9 – 6	9 6 –
5 * 4	5 4 *
8 / 7	8 7 /

Suponga que los operandos aritméticos son números de una sola cifra.

a) Escriba la función **infix** que reciba una cadena de 3 caracteres y verifique que está bien escrita en notación infix. La función devuelve 1 si es una cadena con la expresión infix válida y 0 si no lo es.
b) Escriba la función **postfix** que reciba una cadena de 3 caracteres, previamente validada con la función **infix** y cambie la expresión de notación INFIX a POSTFIX.
c) Escriba un programa de prueba para verificar que las funciónes producen resultados correctos.

11. Escriba una función **strsubpos(c,t)** que entregue la posición inicial de todas las ocurrencias de una subcadena **t** en una cadena de caracteres **c**, como se muestra en el ejemplo:

```
>>> from strsubpos import strsubpos
>>> c='Cada proyecto tiene programas y compromisos'
>>> t='pro'
>>> v=strsubpos(c,t)
>>> v
[6,  21,  36]
```

12. Escriba un programa que lea una lista de palabras y encuentre los anagramas existentes en la lista. Dos palabras son anagramas si contienen las mismas letras aunque estén en orden diferente. Ejemplo. 'roma' y 'mora'

13. El siguiente cuadro muestra de manera divertida la pronunciación de cada letra del alfabeto en lenguaje japonés



Ejemplo. El nombre **JUAN PEREZ** se pronunciaría **zudokato nokushikura**

Escriba un programa que lea el nombre de una persona en lenguaje español y muestre una línea de texto con la pronunciación en el lenguaje japonés. Use el cuadro anterior, como se muestra en el ejemplo.

7.9 Operaciones con matrices

En la notación del lenguaje Python una matriz o arreglo de dos dimensiones es una lista cuyos elementos son listas que tienen la misma longitud y contienen elementos del mismo tipo (usualmente numérico). Estos objetos, igual que los vectores, son fundamentales en las aplicaciones matemáticas y de ingeniería.

Existen módulos o librerías especiales para extender el manejo de vectores y matrices. El más conocido es la librería numérica **NumPy** que será revisada más adelante.

Se puede asociar una matriz a un cuadro con datos organizados en filas y columnas.

Las filas son horizontales y las columnas son verticales, numeradas con índices que comienzan en cero.

Ejemplos. Escribir en la notación de listas de Python la matriz

$$a = \begin{bmatrix} 23 & 45 & 63 \\ 72 & 81 & 91 \\ 56 & 64 & 37 \\ 34 & 75 & 26 \end{bmatrix}$$

```
>>> a=[[23,45,63],[72,81,91],[56,64,37],[34,75,26]]
```
Es una lista de listas

Su representación conceptual es un cuadro en dos dimensiones con 4 filas y 3 columnas. Las filas son horizontales y las columnas son verticales numeradas desde 0

0	23	45	63
1	72	81	91
2	56	64	37
3	34	75	26
	0	1	2

Se pueden manejar filas, columnas o componentes

```
>>> a
[[23, 45, 63], [72, 81, 91], [56, 64, 37], [34, 75, 26]]
```
La matriz almacenada es lineal

```
>>> a[1]
[72, 81, 91]
```
Fila 1

```
>>> for i in range(4):
        print(a[i][1])
45
81
64
75
```
Columna 1

```
>>> a[1][2]
91
```
Componente ubicado en la fila 1, columna 2

```
>>> for i in range(len(a)):              Se puede imprimir para visualizarla por filas
        print(a[i])

[23, 45, 63]
[72, 81, 91]
[56, 64, 37]
[34, 75, 26]
```

7.9.1 Construcción declarativa de listas numéricas (matrices)

Las listas numéricas pueden construirse mediante declaraciones en la ventana interactiva o dentro de programas

a) Mediante declaraciones implícitas

```
>>> c=[[i] for i in range(5)]                          Crea una lista de listas
>>> c
[[0], [1], [2], [3], [4]]

>>> c=[[i,1] for i in range(5)]                        Filas de dos componentes
>>> c
[[0, 1], [1, 1], [2, 1], [3, 1], [4, 1]]

>>> c=[[i,i+1,i+2] for i in range(5)]                  Filas de tres componentes
>>> c
[[0, 1, 2], [1, 2, 3], [2, 3, 4], [3, 4, 5], [4, 5, 6]]
```

a) Mediante declaraciones explícitas

```
>>> a=[]
>>> for i in range(5):                     Crear una matriz con listas vacías (filas)
        a=a+[[]]

>>> a
[[], [], [], [], []]                       Contiene 5 listas vacías (filas de la matriz)
```

Llenar las filas de una matriz 4x3 con enteros consecutivos

```
>>> a=[]                                   Inicia la matriz
>>> for i in range(4):
        f=[]                               Inicia una fila
        for j in range(3):
            f=f+[i+j]                      Llena la fila
        a=a+[f]                            Agrega la fila a la matriz
>>> a
[[0, 1, 2], [1, 2, 3], [2, 3, 4], [3, 4, 5]]
```

Llenar una matriz 4x3 con números aleatorios enteros de una cifra

```
>>> from random import*
>>> a=[]                                    Inicia la matriz
>>> for i in range(4):
      f=[]                                  Inicia una fila
      for j in range(3):
            f=f+[randint(0,9)]              Llena la fila
      a=a+[f]                               Agrega la fila a la matriz
>>> a
[[7, 2, 8], [5, 1, 0], [5, 8, 9], [0, 8, 3]]
```

7.9.2 Algunas funciones de la librería NumPy para matrices

NumPy es una librería de soporte para aplicaciones matemáticas, científicas y de ingeniería. **NumPy** y otra librería con el nombre **SciPy** incluyen una gran cantidad de funciones para aplicaciones en álgebra lineal, interpolación, estadística, optimización, etc.

Este módulo maneja una estructura de datos para manejo de vectores y matrices denominada **array** o arreglo, similares a la listas de Python, con la diferencia que los elementos de un arreglo deben ser del mismo tipo, usualmente numéricos.

Para importar la librería y tener acceso a las funciones, se puede usar la declaración típica:

```
from numpy import*
```

Creación de arreglos con NumPy

```
>>> from numpy import*

>>> v=array([3,6,7,2,8])                         Creación de un vector fila
>>> v
array([3, 6, 7, 2, 8])
>>> v=array([[3],[6],[7],[2],[8]])               Creación de un vector columna
>>> v
array([[3],
       [6],
       [7],
       [2],
       [8]])

>>> a=array([[4,2,5],[2,8,4],[6,9,5]])          Creación de una matriz (arreglo 2 d)
>>> a
array([[4, 2, 5],                                NumPy muestra la matriz por filas
       [2, 8, 4],
       [6, 9, 5]])
```

```
>>> a=array([[4,2,5],[2,8,4],[6,9,5]],float)      Se puede especificar el tipo
>>> a
array([[ 4.,  2.,  5.],
       [ 2.,  8.,  4.],
       [ 6.,  9.,  5.]])
```

Los componentes de las matrices pueden ser manipulados con índices, igual que las listas

```
>>> a[1][2]
4.0
```

Los arreglos admiten también la notación de índices separados por comas. Las listas no.

```
>>> a[1,2]
4.0
```

Se pueden manejar filas o columnas completas

```
>>> a[1,:]
array([ 2.,  8.,  4.])                 Es la fila 1

>>> a[:,1]
array([ 2.,  8.,  9.])                 Contiene los elementos de la columna 1

>>> [n,m]=a.shape                      Dimensiones de un arreglo
>>> n                                  Número de filas
3
>>> m                                  Número de columnas
3
>>> 8 in a                             Operador de pertenencia
True

>>> b=array(range(9),int)              Un arreglo se puede generar con rango
>>> b
array([0, 1, 2, 3, 4, 5, 6, 7, 8])

>>> b=b.reshape((3,3))                 Convertir a un arreglo de 2 dimensiones
>>> b
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])

>>> c=b.tolist()                       Un arrreglo se puede reconvertir a lista
>>> c
[[0, 1, 2], [3, 4, 5], [6, 7, 8]]

>>> v=[[5,4],[7,3]]                    Una lista se puede convertir a arreglo
>>> u=array(v)
>>> u
array([[5, 4],
       [7, 3]])
```

```
>>> b.fill(1)                 Un arreglo se puede rellenar con algún valor
>>> b
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]])
```

7.9.3 Operaciones matemáticas de NumPy con matrices

```
>>> a=[[2,3],[4,5]]           Es una lista
>>> b=[[5,2],[1,4]]           Es una lista
>>> a+b                       + concatena listas
[[2, 3], [4, 5], [5, 2], [1, 4]]

>>> a=array(a)                Conversión a arreglo (matriz)
>>> b=array(b)                Conversión a arreglo (matriz)

>>> a+b                       Suma de matrices
array([[7, 5],
       [5, 9]])
>>> a-b                       Resta de matrices
array([[-3,  1],
       [ 3,  1]])
>>> a*b                       Multiplicación (elemento con elemento)
array([[10,  6],
       [ 4, 20]])
>>> dot(a,b)                  Multiplicación de matrices
array([[13, 16],
       [25, 28]])
```

7.9.4 Algunas funciones especiales de NumPy con matrices

```
>>> a=array([[4,2,5],[2,8,4],[6,9,5]])
>>> a
array([[4, 2, 5],
       [2, 8, 4],
       [6, 9, 5]])

>>> sum(a)                    Suma de elementos de la matriz
45
>>> prod(a)                   Producto de todos los elementos
691200
>>> mean(a)                   Media aritmética de todos los elementos
5.0
>>> std(a)                    Desviación estándar
2.2607766610417559
```

```
>>> sort(a)                                    Ordenamiento por filas
array([[2, 4, 5],
       [2, 4, 8],
       [5, 6, 9]])
>>> diagonal(a)                                Elementos de la diagonal
array([4, 8, 5])

>>> linalg.det(a)                              Determinante
-105.99999999999997

>>> linalg.inv(a)                              Matriz inversa
array([[-0.03773585, -0.33018868,  0.30188679],
       [-0.13207547,  0.09433962,  0.05660377],
       [ 0.28301887,  0.22641509, -0.26415094]])

>>> transpose(a)                               Matriz transpuesta
array([[4, 2, 6],
       [2, 8, 9],
       [5, 4, 5]])

>>> tril(a)                                    Matriz triangular inferior
array([[4, 0, 0],
       [2, 8, 0],
       [6, 9, 5]])

>>> triu(a)                                    Matriz triangular superior
array([[4, 2, 5],
       [0, 8, 4],
       [0, 0, 5]])

>>> a=zeros([3,4],int)                 Llenar una matriz con ceros, tipo entero
>>> a
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])

>>> identity(5)                                Matriz identidad
array([[ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  1.]])
>>> identity(5,int)                            Matriz identidad con enteros
array([[1, 0, 0, 0, 0],
       [0, 1, 0, 0, 0],
       [0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0],
       [0, 0, 0, 0, 1]])
```

La librería **NumPy** fué cargada con la instrucción

```
from numpy import*
```

Esta declaración permite usar las funciones de forma directa y simple como en el ejemplo:

```
>>> a=array([[4,2,5],[2,8,4],[6,9,5]])
>>> sum(a)
45
```

Un inconveniente con esta notación es que en ocasiones entra en conflicto con otras funciones con mismo nombre de la librería estándar de Python. Por ejemplo si se escribe:

```
>>> max(a)
```

Se producirá un error pues la función **max** actúa de manera diferente en las dos librerías.

Debido a esto, muchos usuarios de **Python** y **NumPy** prefieren cargar las librerías asignando una identificación como se muestra en la siguiente instrucción:

```
import numpy as np
```

Ahora, todas las definiciones y funciones se escriben con la notación **np.** para referirse a la librería Numpy y evitar conflictos con funciones con el mismo nombre en otras librerías:

```
>>> a=np.array([[4,2,5],[2,8,4],[6,9,5]])
>>> np.sum(a)
45
```

Esta notación permite usar otras funciones especiales que pertenecen a NumPy

```
>>> np.min(a)                El menor valor del arreglo  a
2
>>> np.argmin(a)             índice, contado por filas
1
>>> np.max(a)                El mayor valor del arreglo a
9
>>> np.argmax(a)             índice, contado por filas
7
```

Aplicación: Resolución de un sistema de ecuaciones lineales con NumPy

Fundamento matemático:

Resolver el sistema

AX = B
A: matriz de coeficientes
B: vector de constantes
X: vector solución

Entonces

$X = A^{-1}B$, si $|A| \neq 0$
A^{-1}: matriz inversa de **A**
|A|: determinante de **A**

Ejemplo. Resolver el sistema

$$\begin{bmatrix} 2 & 4 & 5 \\ 3 & 1 & 4 \\ 5 & 2 & 4 \end{bmatrix} X = \begin{bmatrix} 5 \\ 6 \\ 7 \end{bmatrix}$$

```
>>> from numpy import*
>>> a=array([[2,4,5],[3,1,4],[5,2,4]])          Matriz de coeficientes
>>> b=array([[5],[6],[7]])                      Vector de constantes
>>> linalg.det(a)                               Determinante de la matriz
28.999999999999989
>>> c=linalg.inv(a)                             Matriz inversa
>>> x=dot(c,b)                                  Multipilcar c con b
>>> x
array([[ 0.72413793],                           Vector solución
       [-0.44827586],
       [ 1.06896552]])
```

En lugar de usar la inversa, se puede usar directamente el método **solve** de **Numpy:**

```
>>> x=linalg.solve(a,b)
>>> x
array([[ 0.72413793],
       [-0.44827586],
       [ 1.06896552]])
```

Con la opción **set_printoptions(precision=d)** se puede controlar la cantidad de decimales **d** al imprimir los arreglos de la librería **NumPy**

```
>>> set_printoptions(precision=5)               cinco decimales
>>> x=linalg.solve(a,b)
>>> x
array([[ 0.72414],
       [-0.44828],
       [ 1.06897]])
```

7.9.5 Resolución de problemas con matrices

En esta sección se desarrollarán algunas aplicaciones, programas y funciones, con matrices. Este conocimiento será útil para el desarrollo de nuevas aplicaciones.

Ejemplo. Escriba un programa que reciba una matriz y muestre un vector conteniendo las sumas de las filas.

Solución

Variables

a: matriz de nxm enteros
f: cada fila de la matriz
v: vector con las sumas de las filas de la matriz **a**

Los datos ingresarán uno a la vez guiados por un mensaje que indica la posición de la celda que lo recibe. Los datos se agregan en una fila y cada fila es agregada a la matriz.

```
#Suma de filas de una matriz
n=int(input('cuantas filas: '))
m=int(input('cuantas columnas: '))
a=[]                                    #Iniciar la matriz
for i in range(n):
    f=[]                                #Iniciar una fila
    for j in range(m):
        x=int(input('dato para la fila '+str(i)+
                    ' columna '+str(j)+' : '))
        f=f+[x]                         #Llenar la fila
    a=a+[f]                             #Agregar la fila a la matriz
v=[]
for i in range(n):
    s=0
    for j in range(m):
        s=s+a[i][j]
    v=v+[s]
print('Suma de filas: ',v)
```

```
Prueba del programa
>>>
cuantas filas: 3
cuantas columnas: 2
dato para la fila 0 columna 0 : 4
dato para la fila 0 columna 1 : 2
dato para la fila 1 columna 0 : 3
dato para la fila 1 columna 1 : 5
dato para la fila 2 columna 0 : 6
dato para la fila 2 columna 1 : 4
Suma de filas:  [6, 8, 10]
```

Ejemplo. Reescriba el programa anterior mediante una función que reciba la matriz y retorne un vector conteniendo las sumas de las filas.

Solución

Solución

Nombre de la función: **sumat**
Nombre del módulo: **sumat**
Parámetros
a: matriz
Resultado
v: vector con la sumas de las filas

Se usa la función **shape** de la librería **numpy** para determinar la cantidad de filas y columnas de la matriz que entra a la función

```
from numpy import*
def sumat(a):
    [n,m]=shape(a)
    v=[]
    for i in range(n):
        s=0
        for j in range(m):
            s=s+a[i][j]
        v=v+[s]
    return v
```

Prueba de la función desde la ventana interactiva

```
>>> from sumat import*
>>> a=[[4,5,6],[2,8,4],[2,5,9]]
>>> v=sumat(a)
>>> v
[15, 14, 16]
>>>
```

En lugar de usar la función **shape** de **numpy** se pueden determinar las dimensiones de la matriz con la función **len** de la librería estándar de Python:

```
n=len(a)                Cantidad de filas (listas)
m=len(a[0])             Cantidad de elementos de una lista
```

NOTA. En general, es preferible escribir funciones en vez de programas. Las funciones son más generales y constituyen instrumentos computacionales para resolver problemas desde la ventana interactiva o desde programas que llaman a estas funciones.

Ejemplo. Escriba un programa para llenar una matriz **n**x**n** con los coeficientes del triángulo de Pascal:

1					
1	1				
1	2	1			
1	3	3	1		
1	4	6	4	1	
1	5	10	10	5	1

etc.

Solución

A partir de la tercera fila, los elementos intermedios se obtienen sumando los dos elementos cercanos ubicados en la fila anterior

Variables

p: matriz con los coeficientes
n: número de filas

Se usa la función **zeros** de la librería **numpy** para crear la matriz con ceros y poder acceder a cualquier elemento con los índices. También se puede crear esta matriz llenando filas con ceros y agregándolas a la matriz.

```
from numpy import*
n=int(input('cuantas filas: '))
p=zeros([n,n],int)
for i in range(n):
    p[i][0]=1
    p[i][i]=1
for i in range(2,n):
    for j in range(1,i):
        p[i][j]=p[i-1][j-1]+p[i-1][j]
print(p)
```

Prueba del programa

```
>>>
cuantas filas: 8
[[ 1   0   0   0   0   0   0   0]
 [ 1   1   0   0   0   0   0   0]
 [ 1   2   1   0   0   0   0   0]
 [ 1   3   3   1   0   0   0   0]
 [ 1   4   6   4   1   0   0   0]
 [ 1   5  10  10   5   1   0   0]
 [ 1   6  15  20  15   6   1   0]
 [ 1   7  21  35  35  21   7   1]]
```

Ejemplo. Escriba una función que localice un elemento en una matriz

Solución

Nombre de la función: **matbuscar**
Nombre del módulo: **matbuscar**
Parámetros
a: matriz
x: elemento
Resultados
i,j: fila y columna del elemento **x** en la matriz **a**
si **x** no está en la matriz el resultado será: **-1, -1**

Se usa la función **shape** de la librería **numpy** para determinar la cantidad de filas y columnas de la matriz que entra a la función

```
from numpy import *
def matbuscar(a,x):
    [n,m]=shape(a)
    for i in range(n):
        for j in range(m):
            if x==a[i][j]:
                return [i,j]
    return [-1,-1]
```

Prueba de la función en la ventana interactiva

```
>>> from matbuscar import *
>>> a=[[4, 2], [3, 5], [6, 4]]
>>> [i,j]=matbuscar(a,6)
>>> i
2
>>> j
0
>>> [f,c]=matbuscar(a,9)
>>> f
-1
>>> c
-1
```

Ejemplo. Escriba una función que genere una matriz **nxm** con números aleatorios en un rango especificado

Solución

Nombre de la función: **randmat**
Nombre del módulo: **randmat**
Parámetros
n,m: dimeniones de la matriz
a,b: rango para los números aleatorios
Resultado
c: matriz con números aleatorios

Cada fila f se llena con los números aleatorios y luego es agregada a la matriz

```
from random import*
def randmat(n,m,a,b):
    c=[]
    for i in range(n):
        f=[]
        for j in range(m):
            x=randint(a,b)
            f=f+[x]
        c=c+[f]
    return c
```

Prueba de la función en la ventana interactiva

```
>>> from randmat import*
>>> c=randmat(3,4,10,20)
>>> c
[[18, 13, 12, 12], [13, 18, 14, 11], [17, 11, 13, 17]]
>>>
```

Ejemplo. Para diseñar un juego se necesita una matriz que represente un laberinto de tamaño **n**x**m**. Los bordes deben contener 1 y las celdas interiores deben contener 0 o 1 aleatoriamente. Escriba y pruebe una función para generar la matriz.

Solución

Variables

laber: nombre de la función
n,m: dimensiones del laberinto
a: matriz con el laberinto

Se llena cada fila con números aleatorios **0** o **1** y luego se la agrega a la matriz

```
from random import*
def laber(n,m):
    a=[]
    for i in range(n):
        f=[]
        for j in range(m):
            x=randint(0,1)
            f=f+[x]
        a=a+[f]
    for i in range(n):
        a[i][0]=1
        a[i][m-1]=1
    for j in range(m):
        a[0][j]=1
        a[n-1][j]=1
    return a

#Prueba de la función laber
from numpy import*
a=laber(10,12)
b=array(a)       #Para desplegar como tabla
print(b)
```

```
>>>
[[1 1 1 1 1 1 1 1 1 1 1 1]
 [1 0 1 0 1 0 0 0 0 1 1 1]
 [1 1 0 1 1 0 1 1 1 1 1 1]
 [1 1 1 0 1 1 0 0 1 0 0 1]
 [1 1 1 0 0 1 1 1 1 0 1 1]
 [1 0 0 0 0 1 1 0 0 0 0 1]
 [1 1 1 0 0 1 1 1 0 1 1 1]
 [1 1 0 1 0 1 0 1 0 0 1 1]
 [1 0 0 1 1 1 1 1 1 1 0 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]]
>>>
```

Nota: Las instrucciones de prueba se las escribe junto a la función. Esta es una buena práctica para desarrollar funciones. Después de ser probada, se la puede almacenar separadamente con su nombre o dentro de una librería para que pueda ser importada desde un programa o desde la ventana interactiva.

Ejemplo. Diseñar una aplicación para administrar el uso de los casilleros en un club.

Solución

En un progama se usará una matriz para representar los casilleros. Los elementos de la matriz contendrán el código de identificación del usuario. Un casillero libre contendra cero.

El programa incluirá la función **matbuscar** desarrollada anteriormente

Para la interacción se propone un menú con las siguientes opciones:

1) Asignar casillero
2) Devolver casillero
3) Consultar casillero
4) Consultar usuario
5) Salir

```
#Control de los casilleros de un club
from numpy import *
def matbuscar(a,x):
    [n,m]=shape(a)
    for i in range(n):
        for j in range(m):
            if x==a[i][j]:
                return [i,j]
    return [-1,-1]

n=int(input('Cuantas filas: '))
m=int(input('Cuantas columnas: '))
c=zeros([n,m])        #Iniciar la matriz con casilleros vacíos
while True:
    print('1) Asignar casillero')
    print('2) Devolver casillero')
    print('3) Consultar casillero')
    print('4) Consultar usuario')
    print('5) Salir')
    opc=input('Elija una opción: ')
    if opc=='1':
        i=int(input('Cual fila: '))
        j=int(input('Cual columna: '))
        if c[i][j]==0:
            e=int(input('Ingrese el código: '))
            c[i][j]=e
        else:
            print('Casillero ocupado')
```

```
        elif opc=='2':
            i=int(input('Cual fila: '))
            j=int(input('Cual columna: '))
            if c[i][j]==0:
                print('Casillero no está asignado')
            else:
                c[i][j]=0
        elif opc=='3':
            i=int(input('Cual fila: '))
            j=int(input('Cual columna: '))
            if c[i][j]==0:
                print('Casillero no está asignado')
            else:
                print('El casillero está ocupado')
        elif opc=='4':
            x=int(input('Ingrese el código: '))
            [i,j]=matbuscar(c,x)
            if i>=0 and j>=0:
                print('El usuario está en el casillero: ',i,j)
            else:
                print('El usuario no tiene casillero asignado')
        elif opc=='5':
            print('Adiós')
            break
```

Prueba del programa

```
>>>
Cuantas filas: 4
Cuantas columnas: 4
1) Asignar casillero
2) Devolver casillero
3) Consultar casillero
4) Consultar usuario
5) Salir
Elija una opción: 1
Cual fila: 2
Cual columna: 3
Ingrese el código: 123
1) Asignar casillero
2) Devolver casillero
3) Consultar casillero
4) Consultar usuario
5) Salir
Elija una opción: 3
Cual fila: 2
Cual columna: 3
El casillero está ocupado
. . . . . . .
```

7.9.6 Ejercicios con matrices

1. Lea una matriz cuadrada. Descomponga la matriz en tres matrices: submatriz debajo de la diagonal, submatriz diagonal, y submatriz sobre la diagonal. Verifique que la suma de las tres matrices coincide con la matriz original. Muestre las matrices obtenidas

2. Lea una matriz nxm. Para cada fila, muestre el producto de los elementos cuyo valor es un número par.

3. Lea una matriz cuadrada. Muestre la suma de los elementos que no están en las dos diagonales principales.

4. Uno de los pasos que se requieren en los algoritmos para resolver un sistema de ecuaciones lineales consiste en intercambiar las filas de una matriz cuadrada para colocar en la diagonal principal los elementos de mayor magnitud de cada columna.

Escriba un programa que reciba una matriz cuadrada nxn, intercambie las filas desde arriba hacia debajo de tal manera que el elemento de mayor magnitud de cada columna se ubique en la diagonal y sustituya con ceros el resto de la fila hacia la derecha, como se muestra en el ejemplo.

$$\begin{bmatrix} 2 & 7 & 6 \\ 4 & 5 & 3 \\ 9 & 8 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 9 & 0 & 0 \\ 4 & 5 & 3 \\ 2 & 7 & 6 \end{bmatrix} \Rightarrow \begin{bmatrix} 9 & 0 & 0 \\ 2 & 7 & 0 \\ 4 & 5 & 3 \end{bmatrix}$$

5. Lea una matriz nxn, siendo n un dato inicial. Suponga que cada celda contiene un dato (peso en kg.). Determine cuales son las celdas interiores en las cuales el valor del peso es mayor que el promedio de las cuatro celdas ubicadas a sus cuatro lados inmediatos, es decir que no debe considerar las celdas en los bordes.

6. El área de un patio está distribuida en celdas ordenadas en filas y columnas. En cada celda están almacenados paquetes del mismo tipo.
Escriba un programa que lea una matriz cuyo contenido representa la cantidad de paquetes ubicados en cada celda
a) Encuentre el valor promedio de la cantidad de paquetes existentes en todas las celdas.
b) Encuentre cual celda contiene más paquetes.
c) Muestre las coordenadas y la cantidad de paquetes que contiene una celda cuya posición: número de fila y número de columna, son valores elegidos al azar en el programa..

7. Se dice que una matriz es ‘Diagonal dominante’ si en cada fila, el valor del elemento ubicado en la diagonal, es mayor a la cada uno de los otros elementos de esa fila.

Escriba un programa que lea una matriz nxn y determine si es tipo ‘Diagonal dominante’

8. Escriba un solo programa que lea una matriz **nxn** y desarrolle instrucciones que sucesivamente permitan determinar:

a) La suma de los elementos con valor impar de cada columna
b) El valor promedio de los elementos de cada fila
c) El mayor valor y su posición en cada fila de la matriz
d) La cantidad de elementos en cada fila que son mayores al promedio de la fila
e) El producto de los elementos de la diagonal
f) Sustituya cada elemento impar de la matriz con un núm. aleatorio de una cifra
g) La suma de los elementos de la matriz que no pertenecen a la triangular superior
h) Genere un entero aleatorio de 1 cifra. ¿Cuantas veces está en la matriz de **f)** ?
i) Convierta la matriz en un vector, ordene los elementos y muéstre el vector

9. Un cuadrado semi-mágico es una matriz cuadrada conteniendo números tales que la suma las dos diagonales principales producen el mismo resultado.

Ejemplo. Un cuadrado semi-mágico de 4 filas y 4 columnas:

8	1	6	**7**
6	**5**	**7**	3
4	**3**	**2**	1
2	8	9	**4**

Escriba un programa que coloque números enteros aleatorios de una cifra en una matriz de 4 filas y 4 columnas. Repita el ciclo hasta que la matriz sea un cuadrado semi-mágico.

El programa debe mostrar la matriz resultante y la cantidad de intentos que realizó el programa hasta llenar la matriz con éxito.

10. Escriba un programa que coloque números aleatorios de una cifra en los cuatro bordes de una matriz. Después rellene los elementos del interior de la matriz con números aleatorios de una cifra, tales que cada uno sea menor o igual al promedio de todos los elementos en los bordes

Ejemplo. Matriz inicial de 4 x 4 Matriz rellena:

8	1	6	7		8	1	6	7
6			3	promedio: **4.91**	6	**3**	**4**	3
4			1		4	**2**	**3**	1
2	8	9	4		2	8	9	4

Al inicio no interesan los valores que se asignan a los elementos interiores pues serán sustituidos.

El programa debe mostrar la matriz resultante y la cantidad de intentos que realizó el programa hasta llenar la matriz con éxito.

11. Escriba una función **b=cambiar(a,h,k)** que reciba una matriz nxn e intercambie la fila **h** con la fila **k**. La función debe entregar la matriz transformada.

En la ventana interactiva genere una matriz cuadrada con números aleatorios de una cifra. Llame a la función y verifique si el resultado es correcto.

12. Escriba una función **b=diagonales(a)** que reciba una matriz **nxn** e intercambie los elementos de la diagonal principal con los elementos de la otra diagonal.

Ejm. Matriz inicial de 4 x 4 / Matriz modificada

Matriz inicial de 4 x 4				Matriz modificada			
3	2	7	9	**9**	2	7	**3**
6	5	3	7	6	**3**	**5**	7
8	8	1	6	8	**1**	**8**	6
3	5	9	2	**2**	5	9	**3**

En la ventana interactiva genere una matriz cuadrada con números aleatorios de una cifra. Llame a la función y verifique si el resultado es correcto.

13. Escriba una función que reciba una matriz. La función debe debe entregar un vector con la cantidad de elementos pares que contiene cada columna de la matriz

Ejm. Entra $\begin{bmatrix} 3 & 4 & 5 \\ 6 & 1 & 8 \\ 8 & 6 & 3 \\ 7 & 8 & 7 \end{bmatrix}$ sale **[2, 3, 1]**

Escriba un programa que lea una matriz, llame a la función creada y determine cual es la columna con la mayor cantidad de números pares

14. Lea un vector **x** de **n** componentes. Construya una función **d = mvan(x)** que reciba el vector **x** y entregue la matriz **d** según la definición indicada con el ejemplo:

x = [2, 3, 5, 4]

$$d = \begin{bmatrix} 2^3 & 2^2 & 2^1 & 1 \\ 3^3 & 3^2 & 3^1 & 1 \\ 5^3 & 5^2 & 5^1 & 1 \\ 4^3 & 4^2 & 4^1 & 4 \end{bmatrix}$$

15. Escriba un programa para controlar la cantidad de contenedores en un patio. Ingrese como dato la cantidad inicial y ofrezca las siguientes opciones:

1. Salida de contenedores
2. Llegada de contenedores
3. Cantidad actual de contenedores
4. Terminar el control

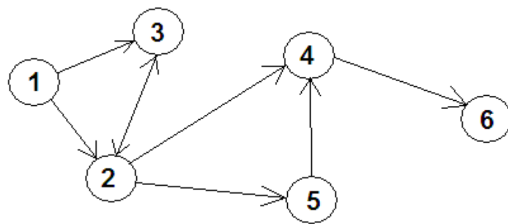
En cada repetición el operador elige la opción ingresando el número y la cantidad de contenedores.

16. Escriba un programa para controlar el uso de los camiones de una empresa. Cada camión tiene un código. Ingrese como dato inicial la lista de los códigos de los camiones. Programe una aplicación con las siguientes opciones:

1. Salida de un camión
2. Devolución de un camión
3. Disponibilidad de un camión
4. Terminar

El operador elige la opción ingresando el número.

17. Un grafo consta de vértices que pueden representarse mediante círculos y arcos que los conectan. Esta conectividad puede describirse mediante una matriz en la que el valor **1** indica que existe un arco en esa dirección, mientras que el valor **0** indica que no existe el arco con esa dirección, como se muestra en el ejemplo



Matriz de conectividad

	1	2	3	4	5	6
1	1	1	1	0	0	0
2	0	1	1	1	1	0
3	0	1	1	0	0	0
4	0	0	0	1	0	1
5	0	0	0	1	1	0
6	0	0	0	1	0	1

Escriba un programa para almacenar 0 o 1 aleatoriamente en una matriz nxn, siendo n un dato que debe leerse inicialmente. Dentro del programa llene la diagonal con 1's para indicar que cada nodo está conectado consigo mismo. El programa examinar las filas para determinar

a) Cual nodo no tiene conecciones con otros nodos (no tiene arcos)

b) Cual es el nodo que tiene más conecciones con otros nodos

18. Versión simple del juego de la vida

Generar una matriz aleatoria nxm con 0’s y 1’s, en donde 1 representa un organismo vivo y 0 su desaparición

Un ciclo de vida significa recorrer todas las celdas de la matriz con las siguientes reglas

a) Si la celda contiene 0 y existe una o dos celdas con 1 en alguno de los cuatro lados, la celda cambia a 1

b) Si la celda contiene 1 y existen tres o cuatro celdas con 1 en sus cuatro lados, su valor cambia a 0

c) En los otros casos, la celda no cambia de valor

Escriba un programa para simular k ciclos de vida. Muestre la cantidad de organismos vivos al inicio y luego de los k ciclos

19. Diseñe un programa para administrar el uso de los casilleros de una institución. Los casilleros están organizados en n filas y m columnas. Inicialmente los casilleros contienen el valor cero, lo cual significa que están vacíos. El programa debe usar un menú con las siguientes opciones:

1) Consultar casillero
2) Asignar casillero
3) Devolver casillero
4) Buscar usuario
5) Salir

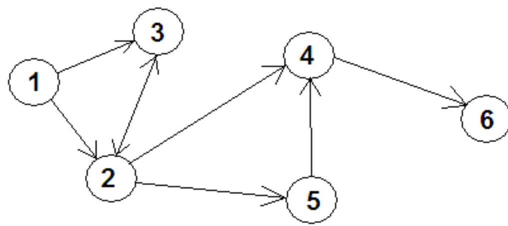
En la opción 1, verificar si el casillero contiene cero, mostrar el mensaje DISPONIBLE u OCUPADO

En la opción 2, almacenar en el casillero el código del usuario asignado

En la opción 3, almacenar cero en el casillero que es devuelto

En la opción 4, ingresar el código de algún usuario. Buscar la ubicación del casillero asignado.

20. Un grafo consta de vértices que pueden representarse mediante círculos y arcos que los conectan. Esta conectividad puede describirse mediante una **matriz** en la que el valor **1** indica que existe un arco en esa dirección, mientras que el valor **0** indica que no existe el arco con esa dirección, como se muestra en el ejemplo



Matriz de conectividad

	1	**2**	**3**	**4**	**5**	**6**
1	1	1	1	0	0	0
2	0	1	1	1	1	0
3	0	1	1	0	0	0
4	0	0	0	1	0	1
5	0	0	0	1	1	0
6	0	0	0	1	0	1

Escriba un programa para manejar interactivamente la conectividad de un grafo mediante un menú con las siguientes opciones:

1) Agregar arco
2) Eliminar arco
3) Consultar arco
4) Listar arcos
5) Salir

Al inicio debe pedir el número de vértices. Llenar con **1** la diagonal y **0** en el resto de la matriz

En la opción 1) debe pedir los vértices inicial y final, y colocar **1** en la celda de la matriz ubicada en la fila y columna respectivas.

En la opción 2) debe pedir los vértices inicial y final, y colocar **0** en la celda de la matriz ubicada en la fila y columna respectivas.

En la opción 3) debe pedir los vértices inicial y final, y mostrar un mensaje "Existe arco" o "No existe arco" dependiendo del contenido de la celda de la matriz ubicada en la fila y columna respectivas

En la opción 4) busque para cada fila (vértice inicial) cada columna (vértices finales), que contenga **1.**

NOTA: Las opciones deben ser instrumentadas mediante **funciones**.

8 Registros y archivos

8.1 Definición de registros

Un registro es un dispositivo para contener datos que pueden tener diferente tipo. Los registros normalmente están asociados a dispositivos externos de almacenamiento como discos. Algunos lenguajes de programación tienen un tipo especial de datos para representar registros.

Python no dispone en su librería estándar este tipo de datos, sin embargo, se puede usar una variable de tipo **lista** como un recipiente natural para almacenar un registro puesto que las listas en Python pueden contener componentes de diferente tipo y además pueden modificarse.

Si se requiere manejar varios registros, cada uno de ellos almacenado en una lista, se puede definir una lista de listas.

Ejemplo. Defina la estructura de un registro para almacenar datos de los artículos de una empresa. Cada artículo contendrá el código, la cantidad, el precio y el nombre.

Variables:

`reg:` Lista. Es el contenedor de un registro
`registros:` Lista de listas. Es el contenedor de los registros en memoria

```
>>> registros=[]

>>> reg=[123,20,12.5,'libro']
>>> registros=registros+[reg]

>>> reg=[234,30,2.5,'cuaderno']
>>> registros=registros+[reg]

>>> registros
[[123, 20, 12.5, 'libro'], [234, 30, 2.5, 'cuaderno']]

>>> registros[1]
[234, 30, 2.5, 'cuaderno']

>>> registros[1][2]
2.5

>>> registros[1][3][0:4]
'cuad'
```

8.2 Desarrollo de una aplicación con registros en memoria

En esta sección se desarrollará un ejemplo para almacenar registros utilizando listas en memoria.

Ejemplo. Crear una aplicación para manejar los datos de los artículos de una empresa.

Datos de cada artículo:

- Código de identificación
- Cantidad actual
- Precio
- Nombre

Opciones disponibles

- **1) Ingresar:** Ingreso de un nuevo artículo: código, cantidad, precio y nombre
- **2) Consultar:** Conocer los datos de un artículo dado su código
- **3) Agregar:** Agregar cantidad a un artículo existente
- **4) Vender:** Vender una cantidad de un artículo existente
- **5) Eliminar:** Eliminar o dar de baja un artículo
- **6) Salir**

Solución

Variables:

registros: Es una lista cuyos componentes son listas que las consideraremos registros para almacenar los datos de los artículos. Esta lista será creada y estará disponible únicamente en la memoria.

Componentes del registro

- **cod:** código del artículo
- **cant:** cantidad actual de artículos disponibles
- **pre:** precio del artículo
- **nom:** nombre del artículo

En esta versión se instrumentará la solución escribiendo en un solo programa todas las instrucciones.

```
#Manejo de registros en memoria
registros=[]
while True:
    print()
    print('1) Ingresar artículo')
    print('2) Consultar artículo')
    print('3) Comprar')
    print('4) Vender')
    print('5) Eliminar artículo')
    print('6) Salir')
    opc=input('Elija una opción: ')
    if opc=='1':
        cod=int(input('Ingrese código: '))
        cant=int(input('Ingrese cantidad: '))
        pre=float(input('Ingrese precio: '))
        nom=input('Ingrese nombre: ')
        reg=[cod,cant,pre,nom]
        registros=registros+[reg]

    elif opc=='2':
        c=int(input('Ingrese código: '))
        p=-1;
        for i in range(len(registros)):
            if c==registros[i][0]:
                p=i
                break
        if p<0:
            print('Artículo no existe')
        else:
            print('Cantidad: ',registros[p][1])
            print('Precio: ',registros[p][2])
            print('Nombre: ',registros[p][3])

    elif opc=='3':
        c=int(input('Ingrese código: '))
        p=-1;
        for i in range(len(registros)):
            if c==registros[i][0]:
                p=i
                break
        if p<0:
            print('Artículo no existe')
        else:
            k=int(input('Ingrese la cantidad comprada: '))
            registros[p][1]=registros[p][1]+k
```

```
    elif opc=='4':
        c=int(input('Ingrese código: '))
        p=-1;
        for i in range(len(registros)):
            if c==registros[i][0]:
                p=i
                break
        if p<0:
            print('Artículo no existe')
        else:
            k=int(input('Ingrese la cantidad vendida: '))
            registros[p][1]=registros[p][1]-k

    elif opc=='5':
        c=int(input('Ingrese código: '))
        p=-1;
        for i in range(len(registros)):
            if c==registros[i][0]:
                p=i
                break
        if p<0:
            print('Artículo no existe')
        else:
            del registros[p]

    elif opc=='6':
        print('Adiós')
        break
```

Prueba del programa

```
1) Ingresar artículo
2) Consultar artículo
3) Comprar
4) Vender
5) Eliminar artículo
6) Salir
Elija una opción: 1
Ingrese código: 123
Ingrese cantidad: 10
Ingrese precio: 18.25
Ingrese nombre: Taladro

1) Ingresar artículo
2) Consultar artículo
3) Comprar
4) Vender
5) Eliminar artículo
6) Salir
```

```
Elija una opción: 1
Ingrese código: 234
Ingrese cantidad: 50
Ingrese precio: 3.25
Ingrese nombre: Flexómetro

1) Ingresar artículo
2) Consultar artículo
3) Comprar
4) Vender
5) Eliminar artículo
6) Salir
Elija una opción: 3
Ingrese código: 234
Ingrese la cantidad comprada: 5

1) Ingresar artículo
2) Consultar artículo
3) Comprar
4) Vender
5) Eliminar artículo
6) Salir
Elija una opción: 2
Ingrese código: 234
Cantidad:  55
Precio:  3.25
Nombre:  Flexómetro

1) Ingresar artículo
2) Consultar artículo
3) Comprar
4) Vender
5) Eliminar artículo
6) Salir
Elija una opción: 6
Adiós
```

8.3 Funciones y métodos para manejo de archivos secuenciales en disco

El lenguaje Python provee las instrucciones para manejo de archivos en el disco.
En la forma básica, la información que se puede almacenar en disco son cadenas de texto.

En las siguientes instrucciones se usará la notación:

f: es el nombre de una variable u objeto creado de tipo archivo
n: es una cadena con el nombre del archivo en el disco.
usuamente agregada la extensión **.txt**

Apertura de un archivo

Esta función se usa para crear o abrir el archivo para su uso

```
f=open(n,t)
```

t: es el tipo de operación que se realizará con el archivo:

Tipos de operación

'w' Para crear el archivo
'a' Crear y agregar datos al archivo
'r' Leer datos del archivo
'r+' Leer y escribir en el archivo

Cierre de un archivo

Al finalizar la operación de un archivo, debe cerrarse con la función

```
f.close()
```

Lectura del contenido de un archivo

Lee una linea de texto del archivo hasta encontrar el carácter de fín de linea: **'\n'**
Si no quedan líneas, retorna una línea vacía

```
v=f.readline()
```

En donde **v** es la variable que recibe la linea de texto

Escritura de texto en el archivo

Escribe en el archivo una línea de texto, normalmente finalizada con el carácter de fin de línea **'\n'**

```
f.write(s)
```

s contiene la línea de texto

Detectar la posición actual del dispositivo de lectura del archivo

```
p=f.tell()
```

p contendrá un entero con la posición actual. La posición inicial en el archivo es **0**

Ubicar el dispositivo de lectura del archivo en una posición específicada

```
f.seek(d)
```

d es el desplazamiento contado a partir del inicio que es la posición **0**

8.4 Conversión de registros a líneas de texto para almacenar en archivos secuenciales en el disco

Para almacenar los registros en el archivo, los datos deben convertirse a texto con la función **str** y armar una línea, separándolos con comas y agregando al final la marca de fin de línea **'\n'**. Esta línea es almacenada en el archivo en disco.

Al traer la línea de texto del disco se la convierte en una lista con la función **split**. De esta lista se toman los componentes convirtiéndolos al tipo original.

Este procedimiento se lo describe con el siguiente ejemplo

Almacenar un registro en el archivo 'nuevo'

```
>>> f=open('nuevo.txt','a')                          Abrir archivo para agregar
>>> reg=[123,20,12.5,'libro']
>>> cod=reg[0]
>>> cant=reg[1]
>>> pre=reg[2]
>>> nom=reg[3]
>>> linea=str(cod)+','+str(cant)+','+str(pre)+','+nom+'\n'     Armar la linea
>>> linea
'123,20,12.5,libro\n'
>>> f.write(linea)                                   Escribir la linea en el archivo
>>> f.close()
```

Leer la línea de texto (registro) del disco y reconstruir los datos con el tipo original

```
>>> f=open('nuevo.txt','r')                          Abrir archivo para lectura
>>> linea=f.readline()
>>> s=linea.split(',')                               Separar componentes en una lista
>>> s
['123', '20', '12.5', 'libro\n']
>>> cod=int(s[0])                                    Convertir componentes a su tipo original
>>> cant=int(s[1])
>>> pre=float(s[2])
>>> nom=s[3]
>>> reg=[cod,cant,pre,nom]
>>> reg
[123, 20, 12.5, 'libro\n']
```

La marca **'\n'** es la marca de fin de linea para la lectura desde el disco. También produce un salto de línea al imprimir la línea en la pantalla. Si no se la desea, se puede eliminar del registro.

8.5 Desarrollo de una aplicación con registros y archivos secuenciales en el disco

En esta sección se explora el uso de archivos secuenciales para almacenar líneas de texto. Estos archivos permiten escribir en el disco las lineas de texto en forma secuencial y también leerlas del disco en forma secuencial.

Ejemplo. Instrumentar una aplicación para almacenar en disco y listar los datos de artículos de una empresa mediante registros permanentes en el disco.

Especificaciones

Componentes de cada artículo
- Código del artículo (entero)
- Cantidad de artículos (entero)
- Precio del artículo (real)
- Nombre del artículo (cadena de caracteres)

Opciones
1) Ingresar artículo
2) Listar artículos
3) Salir

Con la opción 1) se agrega un nuevo artículo al archivo
Con la opción 2) se lista en pantalla los artículos almacenados en el disco

Solución

Para almacenar los registros en el archivo los datos ingresados son convertidos a una línea de texto. Al leer esta línea, se la convierte nuevamente al tipo de los datos originales.

Variables

archivo:	Nombre del archivo
arch:	Objeto tipo archivo para manejo interno
línea:	Linea de texto con los datos separados por comas y la marca de fín de linea
reg:	lista con los componentes de la línea de texto
cod:	Código del artículo (entero)
cant:	Cantidad de artículos (entero)
pre:	Precio del artículo (real)
nom:	Nombre del artículo (cadena de caracteres)

```
from sys import*
nombre=input('Ingrese el nombre del archivo: ')
try:
    arch=open(nombre+'.txt','r')
except FileNotFoundError:
    print('El archivo no existe')
    r=input('Digite  1  si desea crear el archivo: ')
    if r=='1':
        arch=open(nombre+'.txt','w')
        arch.close()
    else:
        exit(0)

while True:
    print('\n')
    print('1) Ingresar artículo')
    print('2) Lista de artículos')
    print('3) Salir')
    opc=input('Elija una opción: ')
    if opc=='1':
        arch=open(nombre+'.txt','a')
        cod=int(input('Ingrese código:   '))
        cant=int(input('Ingrese cantidad: '))
        pre=float(input('Ingrese precio:   '))
        nom=input('Ingrese nombre:   ')
        linea=str(cod)+','+str(cant)+','+str(pre)+','+nom+'\n'
        arch.write(linea)
        arch.close()
    elif opc=='2':
        arch=open(nombre+'.txt','r')
        linea=arch.readline()
        while linea!='':
            reg=linea.split(',')
            print('Código:   ',int(reg[0]))
            print('Cantidad: ',int(reg[1]))
            print('Precio:   ',float(reg[2]))
            print('Nombre:   ',reg[3])
            linea=arch.readline()
        arch.close()
    elif opc=='3':
        print('Adiós')
        break
```

Prueba del programa

```
>>>
Ingrese el nombre del archivo: listart
El archivo no existe
Digite  1  si desea crear el archivo: 1

1) Ingresar artículo
2) Lista de artículos
3) Salir
Elija una opción:  1
Ingrese código:    123
Ingrese cantidad:  20
Ingrese precio:    5.2
Ingrese nombre:    alicate

1) Ingresar artículo
2) Lista de artículos
3) Salir
Elija una opción:  1
Ingrese código:    234
Ingrese cantidad:  40
Ingrese precio:    4.2
Ingrese nombre:    destornillador

1) Ingresar artículo
2) Lista de artículos
3) Salir
Elija una opción:  1
Ingrese código:    345
Ingrese cantidad:  30
Ingrese precio:    3.2
Ingrese nombre:    martillo

1) Ingresar artículo
2) Lista de artículos
3) Salir
Elija una opción: 2

Código:    123
Cantidad:  20
Precio:    5.2
Nombre:    alicate

Código:    234
Cantidad:  40
Precio:    4.2
Nombre:    destornillador
```

```
Código:    345
Cantidad:  30
Precio:    3.2
Nombre:    martillo

1) Ingresar artículo
2) Lista de artículos
3) Salir
Elija una opción: 3
Adiós
```

8.6 Ejercicios con registros y archivos

1. Escriba un programa para el pago a los n vendedores por comisión de una empresa. Para cada vendedor se deben leer registros con los siguientes datos: Código de identificación, nombre del vendedor, nivel (entero que puede ser 1, 2, 3), y el monto en dinero vendido en el mes. El pago depende del nivel: nivel 1: $400, nivel 2: $500, nivel 3: $600. A este valor hay que agregar el 5% del valor de las ventas realizadas. Adicionalmente hay un bono de $100 al o los vendedores con el mayor valor de ventas. Lea los datos y muestre:

a) El valor que hay que pagar a cada vendedor
b) El monto total que se requiere para pagar a todos los vendedores

Los datos deben almacenarse en listas en memoria.

2. Se tiene una lista de n códigos de artículos (números enteros) y la cantidad disponible de cada uno, y otra lista de m clientes (números enteros) junto con el código del artículo que desea (un solo artículo por cliente) y la cantidad requerida.

Almacene ambas listas en memoria y determine la cantidad total sobrante o faltante de cada artículo para atender las solicitudes de todos los clientes.

3. Escriba un programa para control del registro de los estudiantes para un evento.
El sistema debe incluir las siguientes opciones en un menú:
1) Registrar estudiante
2) Eliminar estudiante
3) Consultar registro de estudiantes
4) Mostrar estudiantes registrados
5) Salir

Los datos serán manejados en una lista en memoria

4. Escriba un programa con un menú para registrar estudiantes en uno de los dos paralelos de una materia mediante las opciones indicadas a continuación. Cada paralelo debe ser representado mediante una lista para manejo en memoria.

1) Registrar
   Lea el numero del paralelo elegido (1 o 2), luego lea el código del estudiante y agréguelo a la lista correspondiente
2) Consultar
   Lea el código del estudiante, búsquelo en las listas y muestre el paralelo en el que está registrado
3) Cambiar
   Lea el código del estudiante. Si está registrado elimínelo de la lista y agréguelo a la otra lista
4) Salir

5. Diseñe y pruebe un sistema para control de los socios de un club mediante el menú:

1) Ingresar datos del socio
2) Consultar datos del socio
3) Salir

Los datos deben ser almacenados en un archivo secuencial en disco

Cada socio es un registro con tres datos:

Código: entero, es la identificación del socio.
Categoría: entero: 1 si es niño, 2 si es adulto y 3 si es tercera edad
Cuotas: número de cuotas que el socio adeuda al club.

La primera opción permite ingresar los datos del socio en un registro y almacenarlo en un archivo

La segunda opción pide el código del socio, lo busca en el archivo, y en caso de existir, muestra los datos

6. El Ministerio de Salud requiere implementar un programa para gestión de donantes de sangre que permita registrar y consultar resultados con el menú mostrado. Los datos deben almacenarse en forma permanente en el disco.

1. Ingreso de donante
2. Consulta de donante
3. Cosulta por tipo de sangre
4. Salir

La información que se registra por paciente es: cédula, nombre, edad y tipo de sangre
La consulta por donante muestra el nombre del donante y su tipo de sangre, dado el número de cédula.
La consulta por tipo de sangre presenta el número de donantes por tipo de sangre

El tipo de sangre es un número (1, 2, 3, 4, 5, 6, 7, 8) los cuales corresponden a los siguientes tipos: (1) O-, (2) O+, (3) A-, (4) A+, (5) B-, (6) B+, (7) AB-, (8) AB+

9 Programación modular

El desarrollo de programas computacionales para resolver problemas grandes o complejos no debe intentarse con la escritura de un solo programa. Una técnica adecuada consiste en dividir el problema en subproblemas que puedan ser resueltos separadamente mediante la instrumentación computacional de módulos para resolver cada subproblema. Esta técnica se denomina Programación Modular.

La Programación Modular consiste en escribir funciónes o módulos que puedan ser probados individualmente y finalmente ser integrados. Esta técnica facilita organizar el trabajo en equipos, la localización de errores y la realización de cambios.

9.1 Desarrollo de una aplicación de manejo de registros en memoria

En esta sección se desarrolla una aplicación muy básica usando la **metodología de la programación modular** mencionada anteriormente.

Problema. Diseñar una aplicación interactiva para administrar los registros de estudiantes en un seminario mediante un menú con las siguientes opciones:

1) Agregar registro
2) Eliminar registro
3) Consultar registro
4) Cupo actual
5) Salir

Solución

El programa presentará el menú para que el usuario elija alguna opción.
Cada opción será instrumentada como una función.
El programa llamará a cada función para realizar la acción solicitada.
El programa y las funciones se almacenarán en un solo módulo

Variable

e: vector que contendrá el número de identificación de los estudiantes inscritos
x: cada dato
opc: contiene la opción elegida

Módulos

agregar
eliminar
consultar
cupo

Corresponden a las acciones necesarias para realizar cada opción del menú

Los módulos o funciones pueden escribirse separadamente y almacenarse en una librería o pueden escribirse juntos en el mismo módulo que contiene al programa principal. Esta es la manera que se usa en este ejemplo.

```
def agregar(e):
    x=int(input('Ingrese matrícula: '))
    if x not in e:
        e=e+[x]
    else:
        print('Ya está inscrito')
    return e

def eliminar(e):
    x=int(input('Ingrese matrícula: '))
    if x in e:
        e.remove(x)
    else:
        print('No está inscrito')
    return e

def consultar(e):
    x=int(input('Ingrese matrícula: '))
    if x in e:
        print('Si está inscrito')
    else:
        print('No está inscrito')

def cupo(e):
    n=len(e)
    print('Cantidad de inscritos: ',n)

#Programa principal
e=[]
while True:
    print('1) Agregar registro')
    print('2) Eliminar registro')
    print('3) Consultar registro')
    print('4) Cupo actual')
    print('5) Salir')
    opc=input('Elija una opción: ')
    if opc=='1':
        e=agregar(e)
    elif opc=='2':
        e=eliminar(e)
    elif opc=='3':
        consultar(e)
    elif opc=='4':
        cupo(e)
    elif opc=='5'
        print('Adiós')
        break
```

Prueba del programa

```
>>>
1) Agregar registro
2) Eliminar registro
3) Consultar registro
4) Cupo actual
5) Salir
Elija una opción: 1
Ingrese matrícula: 123
1) Agregar registro
2) Eliminar registro
3) Consultar registro
4) Cupo actual
5) Salir
Elija una opción: 1
Ingrese matrícula: 234
1) Agregar registro
2) Eliminar registro
3) Consultar registro
4) Cupo actual
5) Salir
Elija una opción: 4
Cantidad de inscritos:  2
1) Agregar registro
2) Eliminar registro
3) Consultar registro
4) Cupo actual
5) Salir
Elija una opción: 2
Ingrese matrícula: 123
1) Agregar registro
2) Eliminar registro
3) Consultar registro
4) Cupo actual
5) Salir
Elija una opción: 3
Ingrese matrícula: 123
No está inscrito
1) Agregar registro
2) Eliminar registro
3) Consultar registro
4) Cupo actual
5) Salir
Elija una opción: 5
Adiós
```

9.2 Desarrollo de una aplicación con acceso directo a registros almacenados en disco

En el siguiente ejemplo se desarrolla una aplicación completa en forma modular para manejo de datos almacenados en el disco en un archivo con acceso directo. Esta aplicación debe ser estudiada como una referencia para este tipo de proyectos.

Ejemplo. Instrumentar una aplicación para manejo del inventario de los artículos de una empresa mediante registros permanentes en el disco.

Especificaciones

Componentes de cada artículo
- Código del artículo (entero en 5 columnas)
- Cantidad de artículos (entero en 6 columnas)
- Precio del artículo (real en 8 columnas)
- Nombre del artículo (cadena de caracteres en 20 columnas)

Opciones
1) Ingresar artículo
2) Consultar artículo
3) Comprar
4) Vender
5) Eliminar
6) Salir

Con la opción 1) se agrega un nuevo artículo al archivo
Con la opción 2) se visualiza en pantalla los datos de un artículo dado su código
Las opciones 2) y 3) permitirán actualizar la cantidad de un artículo dado su código
La opción 4) se usará para eliminar un artículo del archivo

Solución

Para almacenar los registros en el archivo los datos ingresados son convertidos a una línea de texto de tamaño fijo (40 caracteres incluyendo la marca de fín de línea '\n').
Al leer esta línea, se la convierte nuevamente al tipo de los datos originales.

Para acceder a cada línea de manera directa se utilizan las funciones **seek()** y **tell()**. De esta manera se puede leer y volver a escribir la línea en la misma posición en el disco..

Los registros son líneas de texto almacenadas en el disco con una longitud fija (40 caracteres incluyendo la marca de fín de línea '\n'). La estrategia para eliminar un registro será asignar el valor 0 a la clave. Al ingresar un nuevo registro, se buscan registros disponibles en el disco (clave 0). Caso contrario, se lo agrega al final del archivo.

El programa incluye validación de datos mediante control de excepciones

Variables

archivo:	Nombre del archivo (variable global)
arch:	Objeto tipo archivo para manejo interno
línea:	Linea de texto con los datos separados por comas y la marca de fín de linea
cod:	Código del artículo (entero)
cant:	Cantidad de artículos (entero)
pre:	Precio del artículo (real)
nom:	Nombre del artículo (cadena de caracteres)

Módulos

Esta aplicación está desarrollada en base a módulos asociados a las acciones básicas que fueron identificadas en varios niveles. La comunicación entre el programa y los módulos usa una variable global para trasmitir el nombre del archivo.

Módulos de acciones principales

apertura:	Solicita el nombre del archivo para leerlo o crearlo
ingreso:	Valida y agrega al archivo un registro con los datos de un nuevo artículo
consulta:	Dado un código, muestra los datos del artículo almacenado
comprar:	Localiza en el disco y modifica el dato de la cantidad de un artículo
vender:	Localiza en el disco y modifica el dato de la cantidad de un artículo
eliminar:	Descarta del disco el registro de un artículo dado su código

Módulos de soporte

leer_registro:	Trae una linea (registro) del disco en una posición especificada
buscar_registro:	Localiza la posición de un registro en el disco dado el código
buscar_bloque_libre:	Localiza un bloque disponible para grabar un nuevo registro
grabar_registro:	Graba una línea (registro) en el archivo
encera_registro:	Anula un registro en el disco asignando cero a la clave
reemplaza_registro:	Reemplaza un registro con otro con datos modificados
linea_a_registro:	Convierte un registro almacenado en el disco como una línea de texto, al formato de una lista para acceder a los componentes.

En las siguientes páginas se muestran los módulos instrumentados. En el último cuadro está el programa principal que llama a los módulos. Para usar esta aplicación deben escribirse juntos los módulos y el programa y almacenar el conjunto con algún nombre.

```
from sys import*
def apertura():
    global archivo
    archivo=input('Ingrese el nombre del archivo: ')
    try:
        arch=open(archivo+'.txt','r')
        arch.close()
    except FileNotFoundError:
        print('El archivo no existe')
        crear=input('Digite 1 para crear el archivo: ')
        if crear=='1':
            arch=open(archivo+'.txt','w')
            arch.close()
        else:
            exit()


def ingreso():
    global archivo
    try:
        c=int(input('Ingrese código  : '))
    except ValueError:
        print('Dato incorrecto')
        return
    if c<=0:
        print('Código incorrecto')
        return
    [exito,pos]=buscar_registro(c)
    if exito:
        print('Código ya existe')
    else:
        try:
            cant=int(input('Ingrese cantidad: '))
            pre=float(input('Ingrese precio  : '))
            nom=input('Ingrese nombre  : ')
        except ValueError:
            print('Dato incorrecto')
            return
        linea=str(c).rjust(5)+','+str(cant).rjust(6)+
                  ','+str(pre).rjust(8)+','+nom.rjust(20)+'\n'
        grabar_registro(linea)
```

```
def consulta():
    global archivo
    try:
        c=int(input('Ingrese código: '))
    except ValueError:
        print('Código incorrecto')
        return
    [exito,pos]=buscar_registro(c)
    if exito:
        linea=leer_registro(pos)
        [cod,cant,pre,nom]=linea_a_registro(linea)
        print('Código:   ',cod)
        print('Cantidad: ',cant)
        print('Precio:   ',pre)
        print('Nombre:   ',nom.strip())
    else:
        print('Registro no existe')


def comprar():
    global archivo
    try:
        c=int(input('Ingrese código: '))
    except ValueError:
        print('Código incorrecto')
        return
    [exito,pos]=buscar_registro(c)
    if exito:
        try:
            k=int(input('Ingrese la cantidad comprada: '))
        except ValueError:
            print('Dato incorrecto')
            return
        reemplaza_registro(pos,k)
    else:
        print('Registro no existe')
```

```
def vender():
    global archivo
    try:
        c=int(input('Ingrese código: '))
    except ValueError:
        print('Código incorrecto')
        return
    [exito,pos]=buscar_registro(c)
    if exito:
        linea=leer_registro(pos)
        [cod,cant,pre,nom]=linea_a_registro(linea)
        try:
            k=int(input('Ingrese la cantidad vendida: '))
        except ValueError:
            print('Dato incorrecto')
            return
        if k>cant:
            print('Cantidad disponible insuficiente')
            return
        reemplaza_registro(pos,-k)
    else:
        print('Registro no existe')


def eliminar():
    global archivo
    try:
        c=int(input('Ingrese código: '))
    except ValueError:
        print('Código incorrecto')
        return
    [exito,pos]=buscar_registro(c)
    if exito:
        encera_registro(pos)
    else:
        print('Registro no existe')


def leer_registro(pos):
    global archivo
    arch=open(archivo+'.txt','r')
    arch.seek(pos)
    linea=arch.readline()
    return linea
```

```
def buscar_registro(c):
    global archivo
    arch=open(archivo+'.txt','r')
    pos=arch.tell()
    linea=arch.readline()
    exito=False
    while linea!='':
        [cod,cant,pre,nom]=linea_a_registro(linea)
        if c==cod:
            exito=True
            break
        pos=arch.tell()
        linea=arch.readline()
    arch.close()
    return [exito,pos]


def grabar_registro(linea):
    global archivo
    [exito,pos]=buscar_bloque_libre()
    if exito:
        arch=open(archivo+'.txt','r+')
        arch.seek(pos)
        arch.write(linea)
        arch.close()
    else:
        arch=open(archivo+'.txt','a')
        arch.write(linea)
        arch.close()


def buscar_bloque_libre():
    global archivo
    arch=open(archivo+'.txt','r')
    pos=arch.tell()
    linea=arch.readline()
    exito=False
    while linea!='':
        [cod,cant,pre,nom]=linea_a_registro(linea)
        if cod==0:
            exito=True
            break
        pos=arch.tell()
        linea=arch.readline()
    arch.close()
    return [exito,pos]
```

```
def encera_registro(pos):
    global archivo
    linea=leer_registro(pos)
    [cod,cant,pre,nom]=linea_a_registro(linea)
    cod=0
    linea=str(c).rjust(5)+','+str(cant).rjust(6)+
              ','+str(pre).rjust(8)+','+nom.rjust(20)+'\n'
    arch=open(archivo+'.txt','r+')
    arch.seek(pos)
    arch.write(linea)
    arch.close()


def reemplaza_registro(pos,k):
    global archivo
    linea=leer_registro(pos)
    [cod,cant,pre,nom]=linea_a_registro(linea)
    cant=cant+k
    linea=str(c).rjust(5)+','+str(cant).rjust(6)+
              ','+str(pre).rjust(8)+','+nom.rjust(20)+'\n'
    arch=open(archivo+'.txt','r+')
    arch.seek(pos)
    arch.write(linea)
    arch.close()


def linea_a_registro(linea):
    x=linea.split(',')
    cod=int(x[0])
    cant=int(x[1])
    pre=float(x[2])
    nom=x[3][0:len(x[3])-1]
    return [cod,cant,pre,nom]
```

```
#Manejo de registros en disco
apertura()
while True:
    print('')
    print('1) Ingresar artículo')
    print('2) Consultar artículo')
    print('3) Comprar')
    print('4) Vender')
    print('5) Eliminar')
    print('6) salir')
    opc=input('Elija una opción: ')
    if opc=='1':
        ingreso()
    elif opc=='2':
        consulta()
    elif opc=='3':
        comprar()
    elif opc=='4':
        vender()
    elif opc=='5':
        eliminar()
    elif opc=='6':
        print('Adiós')
        break
```

Prueba del programa

```
>>>
Ingrese el nombre del archivo: ferretero
El archivo no existe
Digite 1 si desea crear el archivo en el disco: 1

1) Ingresar artículo
2) Consultar artículo
3) Comprar
4) Vender
5) Eliminar
6) salir
Elija una opción: 1
Ingrese código   : 123
Ingrese cantidad: 20
Ingrese precio    : 18.5
Ingrese nombre : Taladro
```

```
1) Ingresar artículo
2) Consultar artículo
3) Comprar
4) Vender
5) Eliminar
6) salir
Elija una opción: 1
Ingrese código  : 234
Ingrese cantidad: 40
Ingrese precio   : 3.25
Ingrese nombre : Flexómetro

1) Ingresar artículo
2) Consultar artículo
3) Comprar
4) Vender
5) Eliminar
6) salir
Elija una opción: 1
Ingrese código  : 345
Ingrese cantidad: 50
Ingrese precio   : 2.45
Ingrese nombre : Destornillador

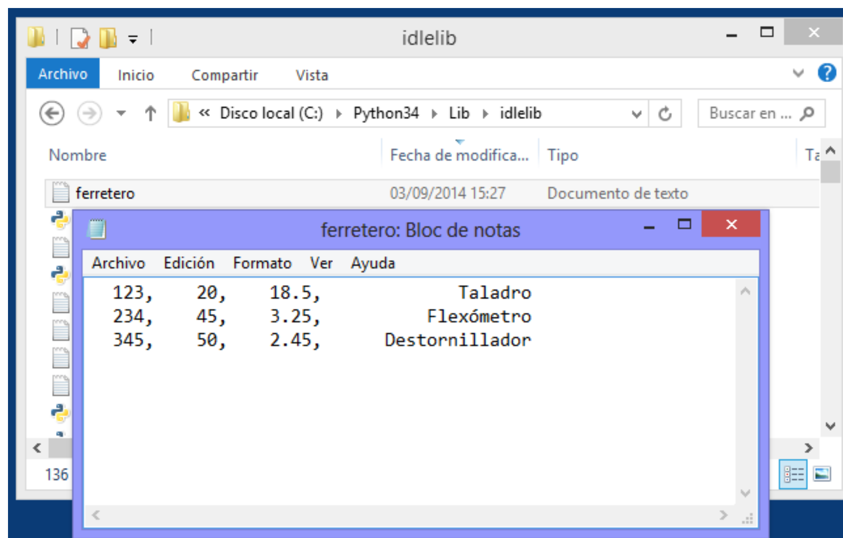
1) Ingresar artículo
2) Consultar artículo
3) Comprar
4) Vender
5) Eliminar
6) salir
Elija una opción: 2
Ingrese código: 234
Código:     234
Cantidad:   40
Precio:      3.25
Nombre:     Flexómetro

1) Ingresar artículo
2) Consultar artículo
3) Comprar
4) Vender
5) Eliminar
6) salir
Elija una opción: 3
Ingrese código: 234
Ingrese la cantidad comprada: 5
```

```
1) Ingresar artículo
2) Consultar artículo
3) Comprar
4) Vender
5) Eliminar
6) salir
Elija una opción: 2
Ingrese código: 234
Código:      234
Cantidad:   45
Precio:       3.25
Nombre:     Flexómetro
```

Los archivos son almacenados en el disco en formato texto, por lo cual se puede visualizar el contenido de estos archivos para constatar y depurar los programas

El siguiente gráfico muestra en pantalla el contenido del archivo que fue creado en el disco para el ejemplo anterior



9.3 Ejercicios de desarrollo de programas de aplicación

1. Diseño de un sistema de registro y control de atención de los pacientes de una clínica.

Los datos deben almacenarse en disco.

Menú con las opciones y acciones que se deben instrumentar

1.- Ingreso de Paciente
    Ingresar y almacenar los datos del paciente:
        Código del paciente
        Código de la enfermedad
        Código del médico tratante
2.- Consulta de paciente
    Ingresar el código del paciente
    Mostrar el código de la enfermedad y el código del médico tratante
3.- Dar de alta a un paciente
    Ingresar el código de un paciente
    Eliminar el registro del paciente
4.- Consulta de médico
    Ingresar el código de un médico
    Mostrar la lista de los códigos de los pacientes asignados
5.- Consulta de enfermedad
    Ingresar el código de una enfermedad
    Mostrar la lista de los códigos de los pacientes que la tienen
6.- Salir

2. Diseño de un sistema para registro y control de alquiler de vehículos

Los datos deben almacenarse en disco.

Menú con las opciones y acciones que se deben instrumentar
1.- Registro de vehículo
    Ingresar y almacenar los datos del vehículo:
        Código del vehículo (entero positivo)
        Tipo de vehículo (1: auto, 2: campero, 3: camioneta)
        Estado del vehículo (0: libre, 1: alquilado, 2: en reparación)
        Código del cliente en uso del vehículo (0 inicialmente)
2.- Consulta de vehículo
    Ingresar el código del vehículo
    Mostrar el tipo y estado del vehículo y el código del cliente en uso.
3.- Alquiler de vehículo
    Leer código del cliente (entero positivo)
    Leer tipo de acción (0: alquila, 1: devuelve)
    Cambiar el estado del vehículo
4.- Reparación de vehículo
    Ingresar el código de un vehículo
    Cambiar el estado del vehículo
5.- Dar de baja vehículo
    Ingresar el código del vehículo
    Eliminar el registro de vehículo del archivo
6.- Salir

3. Diseñe y pruebe un sistema para control del alquiler de los n casilleros de un club. Los casilleros son numerados 1, 2, 3, ..., n.

Inicialmente cada uno contiene el valor 0: disponible

1) Asignar casillero
2) Consultar casillero
3) Buscar usuario
4) Notificar casilleros vencidos
5) Liberar casillero
6) Salir

Cada usuario del casillero contiene los siguientes datos:

Código: entero, es la identificación del socio.
Nombre: Nombre y apellido del socio
Mes: entero que indica el número del mes de vencimiento de uso del casillero

Los datos deben almacenarse en el disco

La opción 1 almacena el código del usuario, nomre y el número del mes de vencimiento.
La opción 2 muestra un mensaje: casillero disponible o el código del usuario y el número del mes de vencimiento de uso, dado el número del casillero
La opción 3, muestra los datos del socio y su casillero asignado, dado el código del socio
La opción 4 lista los casilleros y usuarios cuyo mes es mayor o igual a un mes dado.
La opción 5, libera un casillero dado su número

10 Programación Orientada a Objetos

La metodología de la Programación Orientada a Objetos organiza el desarrollo de la programación usando como centro los datos. Los datos son categorizados mediante clases. Una clase permite empaquetar **atributos** o propiedades que son las variables que recibirán valores y los **métodos** que son funciones que contienen instrucciones.

Cuando se crea una instancia de la clase, esta entidad se denomina **objeto**, el cual tiene acceso a los atributos y a los métodos definidos en la clase.

Hay otros aspectos especializados y que deberán revisarse posteriormente, como la herencia, el polimorfismo, etc.

La clase se la define con la palabra reservada **class**

La clase debe contener un método especial con el nombre **__init__** denominado constructor, el cual inicia algunas variables y ejecuta algunos métodos necesarios

Los métodos deben tener al menos un parámetro, generalmente se acostumbra escribir la palabra **self**, y los demás parámetros requeridos. El parámetro **self** sirve para hacer referencias a los atributos de la clase en cada método.

Los atributos o variables que deben ser accesibles desde fuera de la clase deben declararse anteponiendo la palabra **self.** al nombre de la variable.

Los objetos son instancias o referencias a la clase y deben ser creados antes de acceder a los métodos y atributos.

Ejemplo. Diseñar una clase para describir un artículo con los siguientes atributos: código, cantidad y precio.

Solución

Nombre de la clase: **articulo**

Atributos:	**cod**	(código del artículo)
	cant	(cantidad actual)
	pre	(precio)
Métodos	**cantidad**	(muestra la cantidad actual)
	precio	(muestra el precio)
	vender	(reduce la cantidad actual)
	comprar	(incrementa la cantidad actual)

La clase articulo

```
class articulo():
    def __init__(self,cd,ct,pr):
        self.cod=cd
        self.cant=ct
        self.pre=pr

    def cantidad(self):
        print('Cantidad actual: ',self.cant)

    def precio(self):
        print('Precio: ',self.pre)

    def vender(self,x):
        if x<=self.cant:
            self.cant=self.cant-x
        else:
            print('Cantidad insuficiente')

    def comprar(self,x):
        self.cant=self.cant+x
```

Creación de objetos de la clase artículo en la ventana interactiva

```
>>> from articulo import*
>>> a=articulo(123,20,5.4)            Se crea el objeto  a  de la clase artículo
>>> a.precio()
Precio:   5.4
>>> a.cantidad()
Cantidad actual:   20
>>> a.vender(5)
>>> a.cantidad()
Cantidad actual:   15
>>> b=articulo(234,30,4.2)            Se crea el objeto b  de la clase artículo
>>> b.cantidad()
Cantidad actual:   30
>>> b.comprar(5)
>>> b.cantidad()
Cantidad actual:   35
```

10.1 Diseño de clases para representar estructuras de datos especiales

Las estructuras de datos son dispositivos especiales usados como contenedores de datos. Son diseñados para facilitar las operaciones en ciertas aplicaciones.

El tipo **lista** de Python tiene muchos métodos con los que fácilmente se pueden instrumentar directamente las estructuras de datos lineales básicas: Pila, Cola, Lista. Sin embargo, con fines didácticos y para que el uso de los métodos de Python queden ocultos al programador, se instrumentan las estructuras de datos Pila y Cola usando la metodología de la Programación Orientada a Objetos

10.1.1 Estructura de datos Pila

Una Pila es una estructura lineal que está definida para operar solamente con el dato en el tope o extremo del contenedor de datos que representa a la Pila.

Diseño de la clase Pila

Variable**:**
lista Es el contenedor de datos

Métodos:
Constructor: Inicia el contenedor de datos
Poner: Agrega un elemento al tope de la pila
Sacar: Elimina el elemento del tope de la pila
Tope: Entrega una copia del elemento del tope
Vacía: Detecta si la pila está vacía

```
class pila():
    def __init__(self):            #Constructor
        self.lista=[]              #Inicia el contenedor de datos

    def poner(self,x):             #Agrega un elemento al tope
        self.lista=self.lista+[x]

    def sacar(self):               #Elimina el tope de la pila
        if not self.vacia():
            del self.lista[-1]     #Ubicado como último elemento

    def tope(self):                #Entrega el tope de la pila
        if not self.vacia():
            x=self.lista[-1]
            return x

    def mostrar(self):
        print(self.lista)

    def vacia(self):               #Detecta si la pila está vacía
        return len(self.lista)==0
```

Prueba de la clase pila en la ventana interactiva

```
>>> from pila import *
>>> p=pila()                    Crea el objeto p  de la clase pila
>>> p.poner(3)
>>> p.mostrar()
[3]
>>> p.poner(7)
>>> p.mostrar()
[3, 7]
>>> p.poner(8)
>>> p.mostrar()
[3, 7, 8]
>>> p.sacar()
>>> p.mostrar()
[3, 7]
>>> p.poner(9)
>>> p.mostrar()
[3, 7, 9]
>>> x=p.tope()
>>> x
9
>>> p.mostrar()
[3, 7, 9]
>>> p.vacia()
False

>>> q=pila()                    Crea el objeto q  de la clase pila
>>> q.poner(4)
>>> q.poner(6)
>>> q.mostrar()
[4, 6]
>>> p.mostrar()
[3, 7, 9]
```

Nota. El elemento que se coloca en la pila puede ser un dato estructurado, por ejemplo una lista.

Aplicación de la clase Pila

Búsqueda de la salida en un laberinto

Suponga el problema de encontrar la ruta de salida de un laberinto de **n**x**m** casillas. La casilla inicial es la casilla en la esquina superior izquierda, y la casilla de salida es la casilla en la esquina opuesta. Las otras casillas en los bordes están bloqueadas.

Solución

Las casillas libres se marcan con 0 y las bloqueadas con 1. De cualquier casilla se puede seguir a alguna de las 8 casillas adyacente, siempre que esté libre y no haya sido visitada anteriormente. Se necesita una pila para almacenar la ruta y poder retroceder y continuar con otra casilla libre y no visitada. Una matriz adicional se define para mantener el registro de casillas visitadas.

Variables

q: Matriz que contiene el laberinto
p: Pila que contiene la ruta recorrida en el laberinto
v: Matriz para registrar con 1 las casillas visitadas
dx,dy: Listas con valores para actualizar las coordenadas alrededor de una casilla
x,y: Coordenadas de la ruta

```
from numpy import*
from pila import*

#Definición del laberinto para una prueba
q=array([[0,1,1,1,1,1,1,1],
         [1,0,0,0,1,0,0,1],
         [1,1,1,0,1,1,1,1],
         [1,0,0,1,1,1,1,1],
         [1,0,1,0,1,0,1,1],
         [1,0,1,1,0,1,0,1],
         [1,0,1,0,0,0,0,1],
         [1,1,1,1,1,1,1,0]])

print('Laberinto\n',q)
[n,m]=q.shape

dx=[0,1,1,1,0,-1,-1,-1]    # Valores para actualizar coordenadas
dy=[1,1,0,-1,-1,-1,0,1]
v=zeros([n,m],int)         # Matriz de marcas de casillas visitadas

p=pila()                   # Pila para guardar la ruta recorrida
z=[0,0]                    # Celda inicial
p.poner(z)
salida=False
```

```
while not p.vacia() and not salida:
    [x,y]=p.tope()
    if x==n-1 and y==m-1:      # Si llega a la celda final, salir
        salida=True
        break
    p.sacar()
    nuevo=True
    while nuevo:
        nuevo=False;
        for i in range(8):
            px=x+dx[i]
            py=y+dy[i]
            if px>0 and px<n and py>0 and py<m and
               q[px][py]==0 and v[px][py]==0:
              z=[px,py]        # Colocar celda en la pila
              p.poner(z)       # si hay ruta y no fue visitada
              nuevo=True
              x=px
              y=py
              v[px][py]=1      # Marcar celda visitada
              break

if salida:
  print('Ruta de salida en reversa')
  while not p.vacia():     # Ruta de salida del laberinto en reversa
        [x,y]=p.tope()
        print('x =',x,' y =',y)
        p.sacar()
else:
    print('No hay ruta de salida')
```

Prueba del programa

```
>>>
Laberinto
[[0 1 1 1 1 1 1 1]
 [1 0 0 0 1 0 0 1]
 [1 1 1 0 1 1 1 1]
 [1 0 0 1 1 1 1 1]
 [1 0 1 0 1 0 1 1]
 [1 0 1 1 0 1 0 1]
 [1 0 1 0 0 0 0 1]
 [1 1 1 1 1 1 1 0]]

Ruta de salida en reversa
x = 7  y = 7
x = 6  y = 6
x = 6  y = 5
```

```
x = 5   y = 4
x = 4   y = 3
x = 3   y = 2
x = 2   y = 3
x = 1   y = 3
x = 1   y = 2
x = 1   y = 1
>>>
```

Nota: Para verificar la ruta en la solución, hay que recordar que la numeración de celdas en Python comienza en 0

10.1.2 Estructura de datos Cola

Una Cola es una estructura lineal que está definida para operar con los datos ubicados en el inicio o en el final del contenedor de datos que representa a la cola.

Diseño de la clase Cola

Variable**:**

Lista:	Es el contenedor de datos

Métodos:

Constructor:	Inicia el contenedor
Poner:	Agrega un elemento al final de la cola
Sacar:	Elimina el elemento del frente de la cola
Frente:	Entrega una copia del elemento del frente
Vacía:	Detecta si la cola está vacía

```
class cola():
    def __init__(self):      #Constructor
        self.lista=[]        #Inicia el contenedor de datos vacío

    def poner(self,x):       #Aagrega elemento al final de la cola
        self.lista=[x]+self.lista

    def sacar(self):
        if not self.vacia():
            del self.lista[-1] #Borra el elemento del frente

    def frente(self):           #Entrega una copia del frente
        if not self.vacia():
            x=self.lista[-1]
            return x

    def mostrar(self):
        print(self.lista)

    def vacia(self):            #Detecta si la cola está vacía
        return len(self.lista)==0
```

Prueba de la clase cola en la ventana interactiva

```
>>> from cola import cola
>>> c=cola()
>>> c.poner(34)
>>> c.mostrar()
[34]
>>> c.poner(45)
>>> c.mostrar()
[45, 34]
```

```
>>> c.poner(73)
>>> c.mostrar()
[73, 45, 34]
>>> c.poner(25)
>>> c.mostrar()
[25, 73, 45, 34]
>>> c.sacar()
>>> c.mostrar()
[25, 73, 45]
```

Aplicación de la clase cola

Simulación del comportamiento de una cola de clientes

Ejemplo. Suponer una cola de clientes que serán atendidos en una estación de servicio. Se conoce que la llegada de un cliente en cada minuto tiene distribución uniforme con un valor de probabilidad dado. El tiempo de atención del cliente que está frente a la cola también es un valor aleatorio entero entre 1 y un valor máximo dado como dato en minutos. Se desea conocer cuantos clientes quedarían en la cola luego de trancurrir una cantidad de minutos especificada.

Solución

Variables

- **c:** Cola cuyos elementos contienen el tiempo de atención a cada cliente qyue llega
- **t:** Tiempo de la simulación
- **t_frente:** Tiempo de atención del cliente en el frente de la cola

```
#Simulación de una cola de clientes
from cola import cola
from random import*
t_simul=int(input('Cantidad de minutos para la simulación:  '))
prob_ing=float(input('Probabilidad de ingreso de un cliente en cada
                      minuto:  '))
t_atenc=int(input('Tiempo máximo para atención a cada cliente
                   en minutos:  '))
c=cola()
t=0                 #Tiempo para el proceso de la simulación (reloj)
t_frente=0;         #Tiempo del cliente en el frente
while t<t_simul:
    p=random()           #Probabilidad para la llegada de un cliente
    if p<=prob_ing:
        d=randint(1,t_atenc)    #Duración de la atención del cliente
        c.poner(d)              #Agregar cliente (guardar duración)
    if t_frente<=0:
        if not c.vacia():           #El cliente del frente terminó
            t_frente=c.frente()     #Se inicia el tiempo del frente
            c.sacar()
    else:
        t_frente=t_frente-1     #Restar 1 min al tiempo del frente
    t=t+1;
n=0
while not c.vacia():
    c.sacar()
    n=n+1
print('Cantidad de clientes que quedaron: ',n)
```

Prueba del programa

```
>>>
Cantidad de minutos para la simulación:  120
Probabilidad de ingreso de un cliente en cada minuto:  0.25
Tiempo máximo para atención a cada cliente en minutos:  5
Cantidad de clientes que quedaron: 4
>>>
```

10.2 Ejercicios de Programación Orientada a Objetos

1. Crear una clase Empleado que modele la información que una empresa mantiene sobre cada empleado: código, sueldo base, pago por hora extra, horas extras realizadas en el mes, casado o no y número de hijos.

Al crear objetos de esta clase se deberán proporcionar los datos de un empleado, y los métodos deberán permitir realizar:

a) Cálculo sueldo incluyendo pago de horas extras.
b) Cálculo de retenciones. Suponer 5% si es casado y 5% por cada hijo
c) Cálculo del sueldo neto.
d) Mostrar el detalles de pago

2. Crear una clase Rectangulo para modelar rectángulos por medio de cuatro puntos (los vértices).

Los métodos deberán permitir

a) Trasladar el rectángulo a una posición dados los desplazamientos en cada eje.
b) Contraer las dimensiones, dado un valor en porcentaje.
c) Rotar el rectángulo alrededor del origen, dado el ángulo de rotación
c) Mostrar las coordenadas de los cuatro vértices

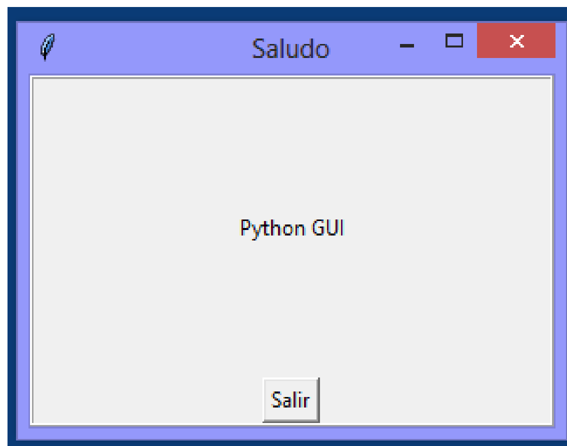
11 Diseño de Interfaz de Usuario

Tkinter es la librería estándar de Python disponible para desarrollar aplicaciones gráficas para interfaz de usuario (GUI).

El uso de los controles disponibles para construir una GUI se puede encontrar en la documentación en línea del shell de Python y en tutoriales en la red.

En el siguiente ejemplo se crea un objeto 'ventana' de la clase Tk de tkinter. Se coloca un mensaje en una etiqueta y se crea un botón para salir de la ventana.

```
from tkinter import*
tk = Tk()
ventana = Frame(tk, relief=RIDGE, borderwidth=2)
ventana.pack(fill=BOTH,expand=1)
tk.title('Saludo')
tk.geometry('300x200')
etiqueta = Label(ventana, text='Python GUI')
etiqueta.pack(fill=X, expand=1)
boton = Button(ventana,text='Salir',command=tk.destroy)
boton.pack(side=BOTTOM)
tk.mainloop()
```



11.1 Diseño de interfaz de usuario con Programación Orientada a Objetos

La Programación Orientada a Objetos provee un procedimiento más formal para definir clases, objetos gráficos, asignación de código y activación.

El siguiente ejemplo es una adaptación del ejemplo que se puede encontrar en la referencia: **Basic Python Tutorial** de **Investary** en el canal de YouTube:

https://www.youtube.com/channel/UCvfluOiZ_eyBfzDXNft_7Eg

Aplicación para crear un botón

```
from tkinter import*
class aplicacion(Frame):

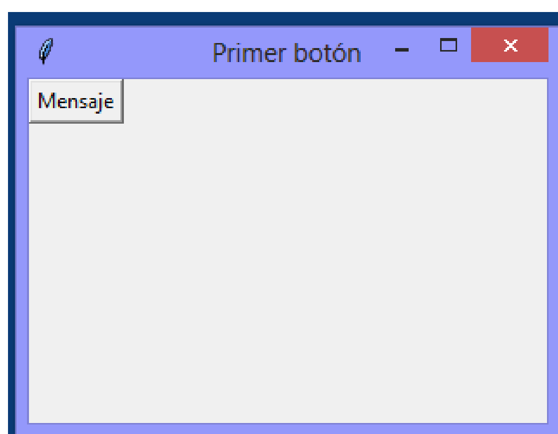
    def __init__(self,master):
        Frame.__init__(self,master)
        self.grid()
        self.crear_widgets()

    def crear_widgets(self):
        self.boton1=Button(self,text="Mensaje")
        self.boton1.grid()

ventana=Tk()
ventana.title('Primer botón')
ventana.geometry('300x200')

app=aplicacion(ventana)

ventana.mainloop()
```



Otras formas de asignar texto al botón

```
from tkinter import*
class aplicacion(Frame):

    def __init__(self,master):
        Frame.__init__(self,master)
        self.grid()
        self.crear_widgets()

    def crear_widgets(self):
        self.boton1=Button(self)
        self.boton1.grid()
        self.boton1.configure(text='Mensaje')


ventana=Tk()
ventana.title('Primer botón')
ventana.geometry('300x200')

app=aplicacion(ventana)

ventana.mainloop()
```

```
from tkinter import*
class aplicacion(Frame):

    def __init__(self,master):
        Frame.__init__(self,master)
        self.grid()
        self.crear_widgets()

    def crear_widgets(self):
        self.boton1=Button(self)
        self.boton1.grid()
        self.boton1['text']='mensaje'


ventana=Tk()
ventana.title('Primer botón')
ventana.geometry('300x200')

app=aplicacion(ventana)

ventana.mainloop()
```

Aplicación para conteo de clicks del botón

```
from tkinter import*
class aplicacion(Frame):

    def __init__(self,master):
        Frame.__init__(self,master)
        self.grid()
        self.boton_clicks=0
        self.crear_widgets()

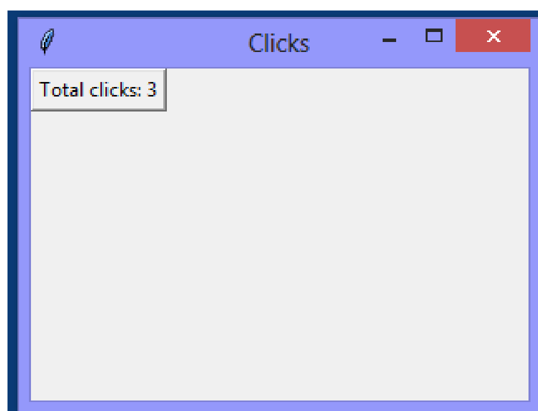
    def crear_widgets(self):
        self.boton1=Button(self)
        self.boton1.grid()
        self.boton1['text']='Conteo de clicks'
        self.boton1['command']=self.actualice_conteo

    def actualice_conteo(self):
        self.boton_clicks=self.boton_clicks+1
        self.boton1['text']='Totalclicks: '+
                     str(self.boton_clicks)

ventana=Tk()
ventana.title('Clicks')
ventana.geometry('300x200')

app=aplicacion(ventana)

ventana.mainloop()
```



Aplicación para ingreso y verificación de un dato

```
from tkinter import*
class aplicacion(Frame):

    def __init__(self,master):
        Frame.__init__(self,master)
        self.grid()
        self.crear_widgets()

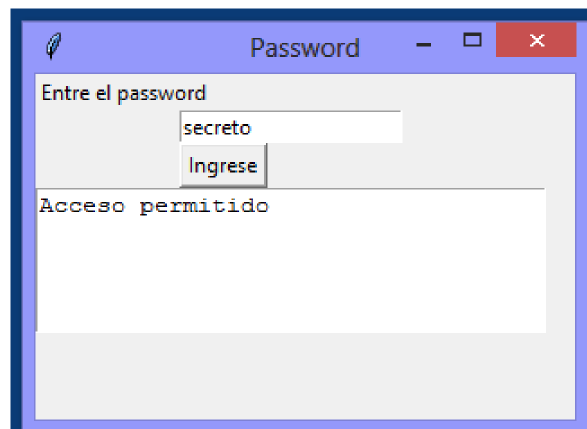
    def crear_widgets(self):
        self.instruction=Label(self,text='Entre el password')
        self.instruction.grid(row=0,column=0,columnspan=2,
                              sticky=W)
        self.password=Entry(self)
        self.password.grid(row=1,column=1,sticky=W)
        self.submit_button=Button(self,text='Ingrese',
                                   command=self.verificar)
        self.submit_button.grid(row=2,column=1,sticky=W)
        self.text=Text(self,width=35,height=5,wrap=WORD)
        self.text.grid(row=3,column=0,columnspan=2,sticky=W)

    def verificar(self):
        contenido=self.password.get()
        if contenido=='secreto':
            mensaje='Acceso permitido'
        else:
            mensaje='Acceso negado'
        self.text.delete(0.0,END)
        self.text.insert(0.0,mensaje)

ventana=Tk()
ventana.title('Password')
ventana.geometry('300x200')

app=aplicacion(ventana)

ventana.mainloop()
```



12 Eficiencia de algoritmos y programas

La eficiencia de un algoritmo y su programación está relacionada con el tiempo necesario para obtener la solución. Este tiempo depende de la cantidad de operaciones que se deben realizar. Así, si se tienen dos algoritmos para resolver un mismo problema, es más eficiente el que requiere menos operaciones para producir el mismo resultado.

Sea **n** el tamaño del problema, y **T(n)** una función que mide la eficiencia del algoritmo (cantidad de operaciones requeridas). Para obtener **T(n)** se pueden realizar pruebas en el computador con diferentes valores de **n** registrando el tiempo real de ejecución. Este tiempo es proporcional a la cantidad de operaciones que se realizaron, por lo tanto se puede usar para estimar la función **T(n)**.

Esta forma experimental para determinar **T(n)** tiene el inconveniente de usar la instrumentación computacional del algoritmo para realizar las pruebas. Es preferible conocer la eficiencia del algoritmo antes de invertir el esfuerzo de la programación computacional para preveer que sea un algoritmo aceptable.

Para determinar la eficiencia de un algoritmo antes de su instrumentación se puede analizar la estructura del algoritmo o realizar un recorrido del mismo en forma abstracta.

Ejemplo. El siguiente algoritmo calcula la suma de los primeros n números naturales. Encontrar T(n)

Sea **T** la cantidad de sumas que se realizan

```
. . .
s = 0
for i in range(n):
   s = s + i
.
```

La suma está dentro de una repetición que se realiza **n** veces, por lo tanto,

T(n) = n

Ejemplo. El siguiente algoritmo suma los elementos de una matriz cuadrada **a** de orden **n**. Encontrar **T(n)**

Sea **T** la cantidad de sumas

```
. . .
s = 0
for i in range(n):
   for j in range(n):
      s = s + a[i][j]
```

La suma está incluida en una repetición doble. La variable **i,** cambia **n** veces y para cada uno de sus valores, la variable **j** cambia **n** veces. Por lo tanto.

T(n) = n^2

Ejemplo. El siguiente algoritmo es una modificación del anterior. Suponga que se desea sumar únicamente los elementos de la sub-matriz triangular superior. Obtener **T(n)**

```
. . .
s = 0
for i in range(n):
   for j in range(i,n):
      s = s + a[i][j]
```

Si no es evidente la forma de **T(n)**, se puede recorrer el algoritmo y anotar la cantidad de sumas que se realizan

Valor	Cantidad de ciclos
i	**j**
0	n
1	n-1
2	n-2
...	**...**
n-1	1

Entonces, **T(n) = 1 + 2 + ... + n =** $\frac{n}{2}(n+1)=\frac{n^2}{2}+\frac{n}{2}$ (suma de una serie aritmética)

Otra manera de obtener la función **T(n)** consiste en programar el conteo de los ciclos de un algoritmo para obtener puntos de su eficiencia.

Ejemplo. Programa para conteo de ciclos para el ejemplo anterior

```
n=int(input('Ingrese n: '))
c=0
for i in range(n):
    for j in range(i,n):
        c=c+1
print(c)
```

Debido a que son dos ciclos, **T(n)** debe ser un polinomio algebraico de segundo grado. Para obtener este polinomio son suficientes tres puntos **(n, c):**

Se realizaron tres pruebas del programa de conteo y se obtuvieron los resultados :

```
>>>
Ingrese n: 4
10

>>>
Ingrese n: 5
15

>>>
Ingrese n: 6
21
```

Con estos resultados se puede construir el polinomio de interpolación que representa a **T(n).** Este polinomio se lo puede obtener manualmente o con un método computacional. En el capítulo 14 se describe el método de Lagange para obtener **T(n)**

El resultado que se obtuvo fue **:**

```
>>> n=[4,5,6]
>>> c=[10,15,21]
>>> p=lagrange(n,c)
t**2/2 + t/2
```

Resultado que coincide con el obtenido anteriormente con un conteo directo manual

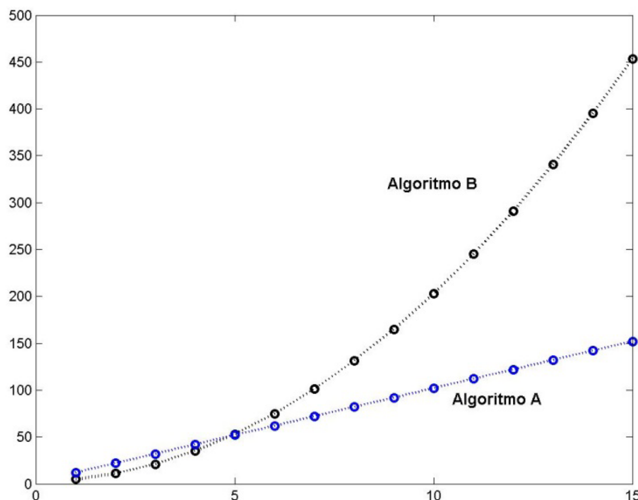
Para medir el tiempo real de ejecución de un proceso (programa o función) se puede usar la función **clock()** de la librería **time**:

```
>>> from time import*
>>> clock(); proceso; clock()
```

12.1 La notación O( )

Supongamos que para resolver un problema se han diseñado dos algoritmos: **A** y **B**, con eficiencias $T_A(n) = 10n+2$, $T_B(n) = 2n^2 + 3$, respectivamente. ¿Cual algoritmo es más eficiente?.

Para valores pequeños de **n**, $T_B(n) < T_A(n)$, pero para valores grandes de **n**, $T_B(n) > T_A(n)$. Es de interés práctico determinar la eficiencia de los algoritmos para valores grandes de **n**, por lo tanto el algoritmo **A** es más eficiente que el algoritmo **B** como se puede observar en el siguiente gráfico:



Si **T(n)** incluye términos de **n** que tienen diferente orden, es suficiente considerar el término de mayor orden pues es el que determina la eficiencia del algoritmo cuando **n** es grande. Para compararlos, no es necesario incluir los coeficientes y las constantes.

Ejemplo. Suponga **$T(n) = n^2 + n + 10$.** Evaluar **T** para algunos valores de **n**

T(2) = 4 + 2 + 10
T(5) = 25 + 5 + 10
T(20) = 400 + 20 + 10
T(100) = 10000 + 100 + 10

Se observa que a medida que **n** crece, **T** depende principalmente del término dominante **n^2.** Este hecho se puede expresar usando la notación **O( )** la cual indica el "orden" de la eficiencia del algoritmo, y se puede escribir: **T(n)** = **$O(n^2)$** lo cual significa que la eficiencia es proporcional a **n^2**, y se dice que el algoritmo tiene eficiencia cuadrática o de segundo orden.

En general, dado un problema de tamaño **n**, la medida de la eficiencia **T(n)** de un algoritmo se puede expresar con la notación **O(g(n))** siendo **g(n)** alguna expresión tal como: **n, n^2, n^3, ..., log(n), n log(n), ..., 2^n, n!, ...** etc, la cual expresa el orden de la cantidad de operaciones que requiere el algoritmo.

Es de interés medir la eficiencia de los algoritmos antes de su instrumentación computacional. En el siguiente cuadro se ha tabulado **T(n)** con algunos valores de **n** y para algunas funciones típicas **g(n)**.

Tabulación de T(n) con algunos valores de n para algunas funciones típicas g(n)

n	[log(n)]	n	[n log(n)]	n^2	n^3	2^n	n!
1	0	1	0	1	1	2	1
3	1	3	3	9	27	8	6
5	1	5	8	25	125	32	120
7	1	7	13	49	343	128	5040
9	2	9	19	81	729	512	3.62×10^5
11	2	11	26	121	1331	2048	3.99×10^7
13	2	13	33	169	2197	8192	6.22×10^9
15	2	15	40	225	3375	32768	1.30×10^{12}
17	2	17	48	289	4913	1.31×10^5	3.55×10^{14}
19	2	19	55	361	6859	5.24×10^5	1.21×10^{17}
21	3	21	63	441	9261	2.09×10^6	5.10×10^{19}
23	3	23	72	529	12167	8.38×10^6	2.58×10^{22}
25	3	25	80	625	15625	3.35×10^7	1.55×10^{25}
50	3	50	195	2500	125000	1.12×10^{15}	3.04×10^{64}
100	4	100	460	10000	1000000	1.26×10^{30}	9.33×10^{157}

Los algoritmos en las dos últimas columnas son de tipo **exponencial** y **factorial** respectivamente. Se puede observar que aún con valores relativamente pequeños de **n** (mayor a 100) el valor **T(n)** es muy alto. Los algoritmos con este tipo de eficiencia se denominan **no factibles** pues ningún computador actual pudiera calcular la solución para valores de **n** grandes en un tiempo aceptable.

.

13 Librerías especializadas

Algunos temas revisados en estas secciones requieren que los usuarios tengan un nivel matemático mayor al nivel básico utilizado en los capítulos anteriores

13.1 Librería gráfica: Pylab, Matplotlib

Esta librería puede decargarse de la dirección:

http://www.lfd.uci.edu/~gohlke/pythonlibs/

Pylab incluye la librería gráfica **Matplotlib** junto con las librerías numéricas **NumPy**, **SciPy**

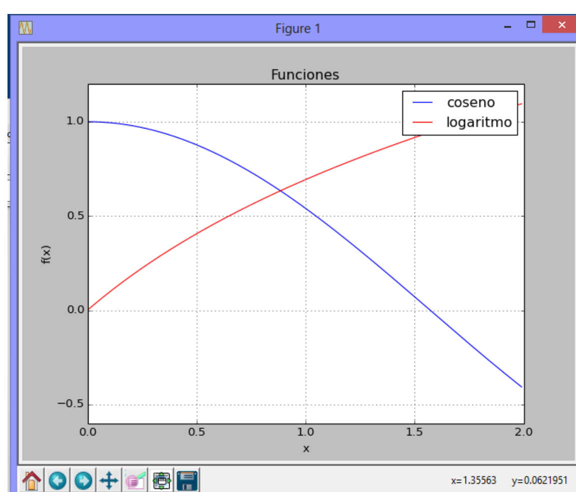
13.1.1 Gráficos en el plano

Algunos códigos y símbolos para graficar

plot: Función para graficar en el plano con el estilo, marca y color especificados:
marcas: **'o'**: círculos, **'.'**: puntos, **'*'**: estrellas, **'s'**: cuadrados,
'+': cruces, **'^'**: triángulos, **'p'**: pentágonos, **'h'**: hexágonos,
'-': línea contínua, **'- -'**: línea discontínua, **':'**: línea punteada
color: **'b'**: blue, **'g'**: green, **'r'**: red, **'k'**: black, **'y'**: yellow, **'c'**: cyan

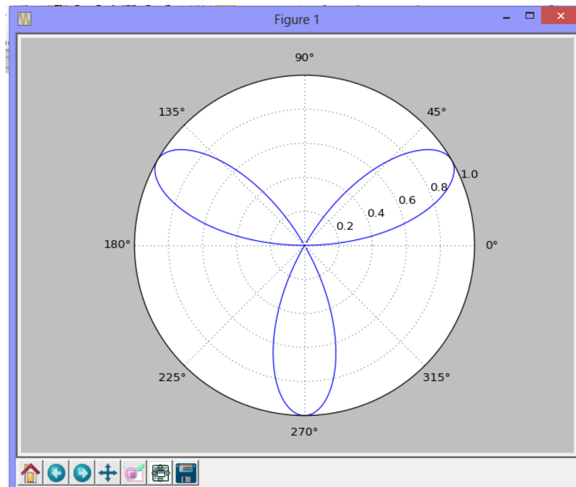
title: Agrega el título
xlabel, ylabel: Coloca nombres a los ejes
grid: Muestra las cuadrículas
legend: Muestra un recuadro con identificación para los gráficos
loc: Ubicación del recuadro: **'lower'**, **'upper'**, **'center'**, **'left'**, **'right'**
show: Despliega el gráfico en pantalla

```
>>> from pylab import*
>>> x=arange(0,2,0.1)                Genera el vector  x=[0, 0.1, 0.2, 0.3, . . . , 2]
>>> plot(x,cos(x),'-b')              Grafica los puntos  (x, cos(x)) con una linea azul
>>> plot(x,log(x+1),'-r')            Grafica los puntos  (x, log(x+1)) con una línea roja
>>> title('Funciones')
>>> xlabel('x')
>>> ylabel('f(x)')
>>> grid(True)
>>> legend(('coseno','logaritmo'),loc='upper right')
>>> show()
```

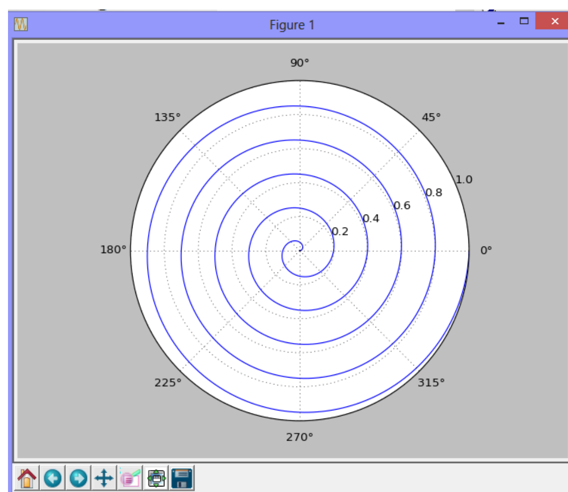


13.1.2 Gráficos en coordenadas polares

```
>>> from pylab import*
>>> t=arange(0,2*pi,0.01)
>>> r=sin(3*t)
>>> polar(t,r)
>>> show()
```

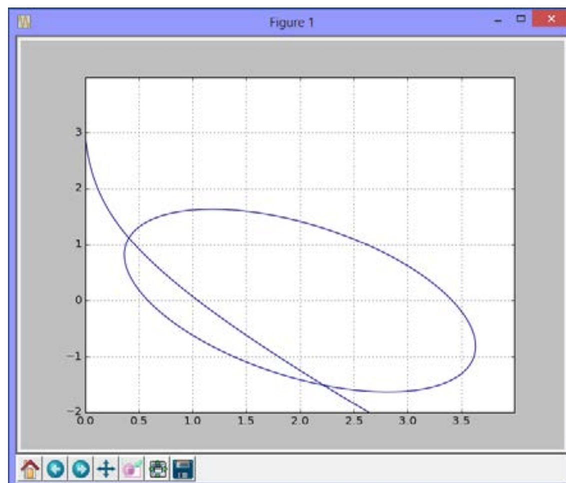



```
>>> from pylab import*
>>> t=arange(0,10*pi,0.01)
>>> r=t/(10*pi)
>>> polar(t,r)
>>> show()
```



13.1.3 Gráficos de ecuaciones implícitas

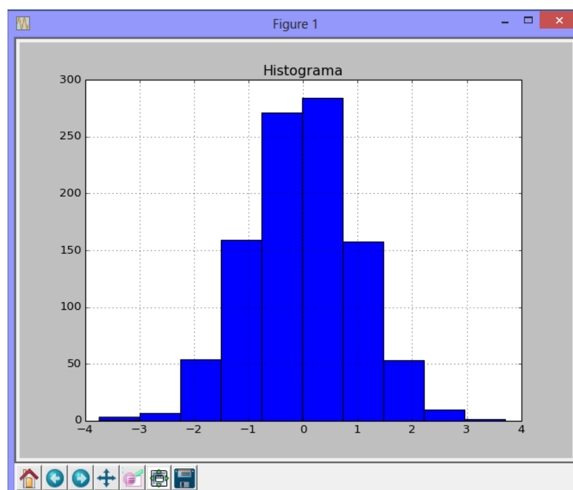
```
>>> from pylab import*
>>> xrango = arange(0,4,0.01)
>>> yrango = arange(-2,4,0.01)
>>> x, y = meshgrid(xrango,yrango)
>>> f=(x-2)**2+(y-1)**2+x*y-3
>>> g=x*exp(x+y)+y-3
>>> contour(x, y, f,[0])
>>> contour(x, y, g,[0])
>>> grid(True)
>>> show()
```



Nota. Las funciones matemáticas del módulo **math** no se pueden usar con **meshgrid**

13.1.4 Graficación de histogramas

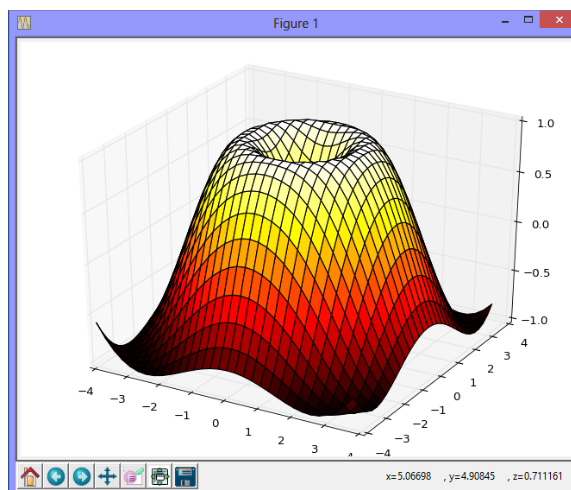
```
>>> from pylab import*
>>> from random import*
>>> n=normal(size=100)                    Una muestra normal de números aleatorios
>>> hist(n)
>>> title('Histograma')
>>> grid(True)
>>> show()
```



13.1.5 Gráficos 3D

Referencia para este ejemplo [6]

```
>>> from pylab import*
>>> from mpl_toolkits.mplot3d import Axes3D
>>> fig=figure()
>>> ax=Axes3D(fig)
>>> x=arange(-4,4,0.25)
>>> y=arange(-4,4,0.25)
>>> X,Y=meshgrid(x,y)
>>> R=sqrt(X**2+Y**2)
>>> Z=sin(R)
>>> ax.plot_surface(X,Y,Z,rstride=1,cstride=1,cmap='hot')
>>> show()
```



13.2 Librería para manejo matemático simbólico: SymPy

Con la librería **SymPy** se puede explorar el manejo matemático simbólico.

Esta librería puede decargarse de la dirección:

http://www.lfd.uci.edu/~gohlke/pythonlibs/

Referencia para algunos ejemplos [7]

Para acceder a esta librería

```
>>> from sympy import*
```

13.2.1 Declaración de variables smbólicas

```
>>> x=Symbol('x')
>>> a,b=symbols('a,b')
```

13.2.2 Operaciones algebraicas

```
>>> 3*x+2*x
5*x
>>> v=[2*x,3*x,5*x]
>>> sum(v)
10*x

>>> g=(2+x)**2
>>> u=expand(g)
>>> u
x**2 + 4*x + 4
>>> factor(u)
(x + 2)**2
```

13.2.3 Evaluación de expresiones

```
>>> u=Symbol('u')
>>> v=sin(u)
>>> v.subs(u,1.2)                    evaluar por sustitución
0.932039085967226
>>> v.subs(u,1.2).evalf(5)           evalf especifica dígitos
0.93204

>>> x,y=symbols('x,y')
>>> f=2*exp(x)+3*y+1
>>> r=f.subs(x,2).subs(y,3)
>>> r
10 + 2*exp(2)
>>> r=f.subs(x,2).subs(y,3).evalf(8)
>>> r
24.778112
```

```
>>> a,b=symbols('a,b')
>>> expand(sin(a+b),trig=True)
sin(a)*cos(b) + sin(b)*cos(a)

>>> simplify(sin(a)**2+cos(a)**2)
1
```

13.2.4 Operaciones del cálculo

```
>>> x,y=symbols('x,y')
>>> f=x*exp(x)+y**2+1
>>> diff(f,x)                               Derivar
x*exp(x) + exp(x)

>>> diff(f,y)
2*y

>>> f=x*exp(x)+x**2+1
>>> diff(f,x).subs(x,3)                     Evaluar derivada
6 + 4*exp(3)

>>> integrate(f,x)                          Integrar
x**3/3 + x + (x - 1)*exp(x)
>>> integrate(f,(x,0,2))                    Evaluar integral
17/3 + exp(2)
>>> integrate(f,(x,0,2)).evalf(8)           8 dígitos
13.055723

>>> f=x*exp(x)+y**2+1
>>> integrate(f,x)
x*(y**2 + 1) + (x - 1)*exp(x)
>>> integrate(f,y)
y**3/3 + y*(x*exp(x) + 1)

>>> integrate(f,(x,0,2),(y,1,3))
2*exp(2) + 70/3

>>> n=Symbol('n')
>>> Sum(1/n**2,(n,1,10)).evalf(8)
1.5497677

>>> limit(sin(x)/x,x,0)
1
>>> limit(1/x,x,oo)
0
>>> limit(1/x,x,0,dir='+')
oo
>>> limit(1/x,x,0,dir='-')
-oo

>>> series(exp(x),x,1,7)
E+E*x+E*x**2/2+E*x**3/6+E*x**4/24+E*x**5/120+E*x**6/720+O(x**7)
```

13.2.5 Resolución de ecuaciones

Este componente de la librería SymPy todavía está en desarrollo

```
>>> from sympy import*
>>> x=Symbol('x')
```

Resolver la ecuación polinómica: $x^3 - 2x - 5.2 = 0$

```
>>> u=solve(x**3-2*x-5.2)
>>> u
[2.11229262318742,
 -1.05614631159371 - 1.16031680780681*I,
 -1.05614631159371 + 1.16031680780681*I]
```

El resultado es un vector
Una raíz real y dos raíces complejas
I es √-1

Resolver la ecuación no lineal: $e^x - \pi x = 0$

```
>>> u=solve(exp(x)-pi*x)
>>> u
[-LambertW(-1/pi)]
>>> float(u[0])
0.5538270366445136
```

El resultado es un vector
Solución simbólica
Solución numérica

Resolver la ecuación no lineal: $\cos(x) - \pi x = 0$

```
>>> u=solve(cos(x)-pi*x)
No algorithms are implemented to solve equation cos(x) - pi*x
```

No encontró la solución

```
>>> from sympy import*
>>> x=Symbol('x')
>>> y=Function('y')
```

Resolver la ecuación diferencial: $y'(x) + x - 1 = 0$

```
>>> dsolve(Derivative(y(x),x)+x-1)
y(x) == C1 - x**2/2 + x
```

Resolver la ecuación diferencial: $y'(x) + y(x) + x - 1 = 0$

```
>>> dsolve(Derivative(y(x),x)+y(x)+x-1)
y(x) == (C1 + (-x + 2)*exp(x))*exp(-x)
```

Resolver la ecuación diferencial: $y''(x) + y'(x) + x - 1 = 0$

```
>>> dsolve(Derivative(y(x),x,x)+Derivative(y(x),x)+x-1)
y(x) == C1 + C2*exp(-x) - x**2/2 + 2*x
```

13.2.6 Salida formateada de expresiones

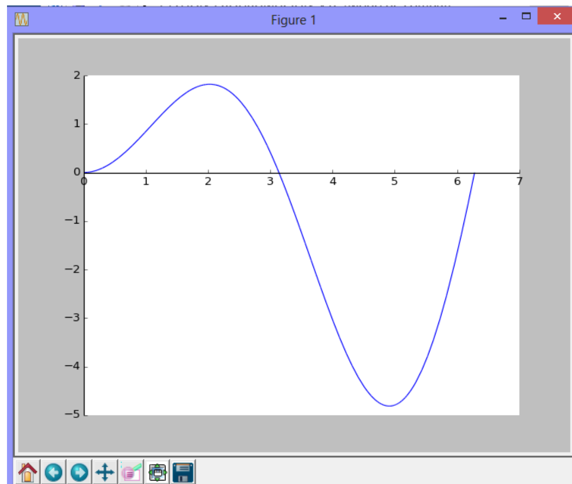
```
>>> from sympy import*
>>> x=Symbol('x')
>>> y=(x+1)**2/(x+3)
>>> pprint(y)
      2
(x + 1)
--------
 x + 3
```

pprint es 'pretty print'

SymPy tiene un módulo para graficar

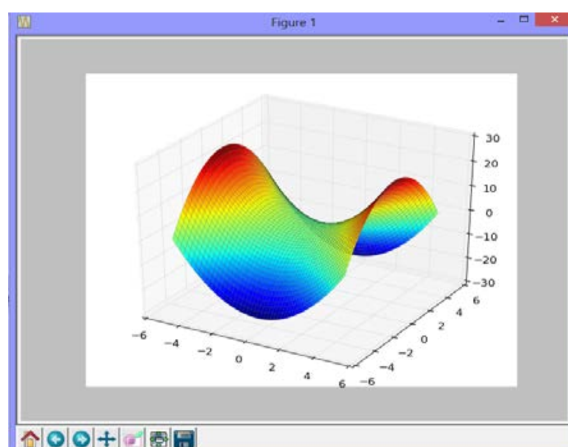
13.2.7 Gráficos en el plano con SymPy

```
>>> from sympy import*
>>> x=Symbol('x')
>>> f=x*sin(x)
>>> plot(f,(x,0,2*pi))
```



13.2.8 Gráficos 3D con SymPy

```
>>> from sympy import*
>>> from sympy.plotting import*
>>> x,y=symbols('x,y')
>>> z=x**2-y**2
>>> plot3d(z,(x,-5,5),(y,-5,5))
```



14 Métodos Numéricos

Los métodos numéricos son alternativas para resolver problemas matemáticos para los cuales no se puede, o es muy laborioso, obtener la solución con métodos analíticos, o si los métodos computacionales disponibles no proporcionan la respuesta correcta.

Esta sección tiene como referencia a [10]

14.1 Resolución de problemas en la ventana interactiva

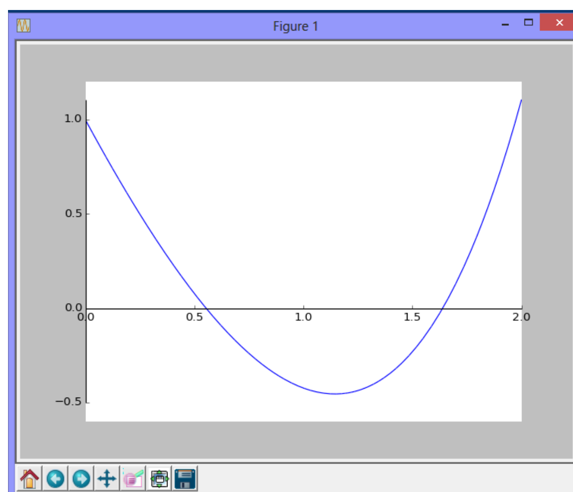
Cálculo de una raíz real con la fórmula de Newton:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}, \quad f'(x_i) \neq 0, \; i = 0, 1, 2, \ldots$$

Ejemplo. Calcular una raíz real de la ecuacion: $f(x) = e^x - \pi x = 0$

Librería de matemáticas simbólicas: **Sympy**

```
>>> from sympy import*
>>> x=Symbol('x')
>>> f=exp(x)-pi*x
>>> plot(f,(x,0,2))
```




```
>>> df=diff(f,x)                                          Obtención de f'(x)
>>> r=0.5                                                 Valor inicial del gráfico
>>> r=r-float(f.subs(x,r))/float(df.subs(x,r))            float evalúa la expresión
>>> r
0.552198029112459
>>> r=r-float(f.subs(x,r))/float(df.subs(x,r))
>>> r
0.5538253947739784
>>> r=r-float(f.subs(x,r))/float(df.subs(x,r))
>>> r
0.5538270366428404
```

```
>>> r=r-float(f.subs(x,r))/float(df.subs(x,r))
>>> r
0.5538270366445136

>>> r=r-float(f.subs(x,r))/float(df.subs(x,r))
>>> r
0.5538270366445136

>>> float(f.subs(x,r))                          Verificar si f(r) = 0
-1.5192266539886956e-17
```

Si se comienza con **r = 1.5** se puede calcular la otra raíz real:

```
>>> r=1.5
. . .
. . .
. . .
>>> r=r-float(f.subs(x,r))/float(df.subs(x,r))
>>> r
1.6385284199703636

>>> float(f.subs(x,r))                          Verificar si f(r) = 0
4.591445985947165e-16
```

Uso de la función solve de Sympy para obtener las raíces de la ecuacion anterior:

```
>>> from sympy import*
>>> x=Symbol('x')
>>> f=exp(x)-pi*x
>>> r=solve(f)
>>> len(r)                 La función solve solo permitió calcular una raíz
1
>>> r
[-LambertW(-1/pi)]         Solución expresada con constantes
>>> float(r[0])
0.5538270366445136         Solución en formato decimal
>>> r[0].evalf(6)
0.553827                   evalf  controla cuantos dígitos
```

14.2 Librería de métodos numéricos

En la siguiente librería se han instrumentado como referencia, algunos métodos numéricos clásicos. También se incluyen ejemplos de su utilización. La formulación matemátca para cada método se la puede encontrar en la referencia bibliográfica [10].

La librería incluye los módulos de los métodos numéricos y puede extenderse con nuevas funciones. Se la almacenó con el nombre **metodos** con el que debe importarse para su aplicación como se indica en los ejemplos.

Los métodos numéricos instrumentados inicialmente son:

Método de la Bisección
Cálculo de raíces reales de una ecuación

Método de Gauss-Jordan
Resolución de un sistema de ecuaciones lineales

Método de interpolación de Lagrange
Obtencion y evaluación del polinomio de interpolación dado un conjunto de puntos

Método de Simpson
Integración numérica de una función acotada de una variable en un intervalo

Método de Runge-Kutta
Resolución de una ecuación diferencial de primer orden con una condición en el inicio.

```python
# Librería de métodos numéricos

# Cálculo de raíces reales de una ecuación: Bisección
def biseccion(f, a, b, e):
    while b-a>=e:
        c=(a+b)/2
        if f(c)==0:
            return c
        else:
            if f(a)*f(c)>0:
                a=c
            else:
                b=c
    return c

#Solución de un sistema de ecuaciones lineales: Gauss-Jordan
def gaussjordan(a,b):
    n=len(b)
    for i in range(n):
        a[i]=a[i]+[b[i]]                          #Matriz aumentada

    for e in range(n):
        p=e
        for i in range(e+1,n):
            if abs(a[i][e])>abs(a[p][e]):         #Buscar pivote
                p=i

        a[e],a[p]=a[p],a[e]                       #Intercambia filas
        t=a[e][e]
        if abs(t)<1e-10:                          #Matriz singular
            return []

        for j in range(e,n+1):
            a[e][j]=a[e][j]/t                     #Normalizar fila

        for i in range(n):
            if i!=e:
                t=a[i][e]
                for j in range(e,n+1):
                    a[i][j]=a[i][j]-t*a[e][j]     #Reducir filas

    x=[]
    for i in range(n):
        x=x+[a[i][n]]                             #Solución
    return x
```

```
#Polinomio de Interpolación: Método de Lagrange
from sympy import *
def lagrange(x,y,u=None):
    n=len(x)
    if u==None:
        t=Symbol('t')
    else:
        t=u
    p=0
    for i in range(0,n):
        L=1
        for j in range(0,n):
            if j!=i:
                L=L*(t-x[j])/(x[i]-x[j])
        p=p+y[i]*L
        p=expand(p)
    return p

# Integración numérica: Fórmula de Simpson
def simpson(f, a, b, m):
    h=(b-a)/m
    s=0
    x=a
    for i in range (1,m):
        if i%2==1:
            s=s+4*f(x+i*h)
        else:
            s=s+2*f(x+i*h)
    s=h/3*(f(a)+s+f(b))
    return s

# Solución de una E.D.O: Método de Runge-Kutta de cuarto orden
def rungekutta(f,x,y,h,m):
    u=[]
    v=[]
    for i in range(m):
        k1=h*f(x,y)
        k2=h*f(x+h/2, y+k1/2)
        k3=h*f(x+h/2, y+k2/2)
        k4=h*f(x+h, y+k3)
        y=y+1/6*(k1+2*k2+2*k3+k4)
        x=x+h
        u=u+[x]
        v=v+[y]
    return [u,v]
```

14.3 Aplicación de los métodos numericos de la librería Métodos

Resolución de ejemplos en la ventana interactiva

Para aplicar los métodos se deben cargar las librerias **Pylab** y **Métodos**

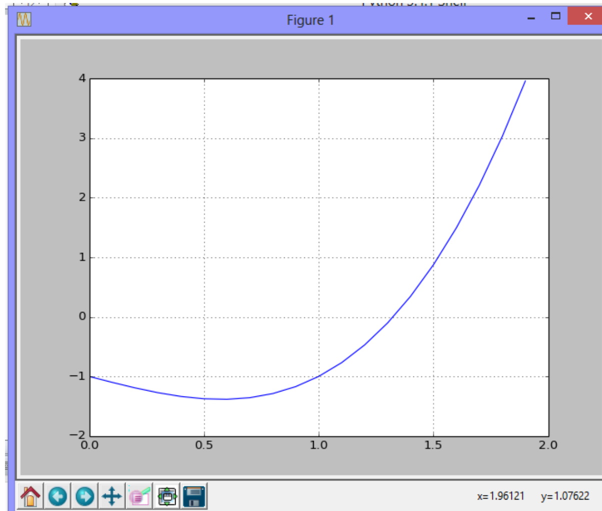
a) Aplicación del método de la Bisección

Resolver la ecuación: **f(x) = x^3 – x – 1 = 0**

```
>>> from pylab import*
>>> from metodos import*

>>> def f(x):return x**3-x-1

>>> t=arange(0,2,0.1)
>>> plot(t,f(t))
>>> grid(True)
>>> show()
```




```
>>> c=biseccion(f,1,2,0.00001)
>>> c
1.3247146606445312
>>> f(c)
-1.40587468702158e-05
```

Raíz calculada con cinco decimales

Verificar en la ecuación

b) Aplicación del método de Gauss-Jordan

Resolver el sistema

$$\begin{bmatrix} 8 & 4 & 3 \\ 2 & 4 & 1 \\ 5 & 7 & 4 \end{bmatrix} X = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$$

```
>>> from pylab import*
>>> from metodos import*

>>> a=[[8,4,3],[2,4,1],[5,7,4]]
>>> b=[2,3,4]
>>> x=gaussjordan(a,b)
>>> x
[0.02380952380952378, 0.8809523809523809, -0.5714285714285714]
```

c) Aplicación del método de interpolación de Lagrange

Colocar un polinomio de interpolación sobre los puntos

(x, f(x)): (2, 4), (3, 5), (5, 7), (6, 6)

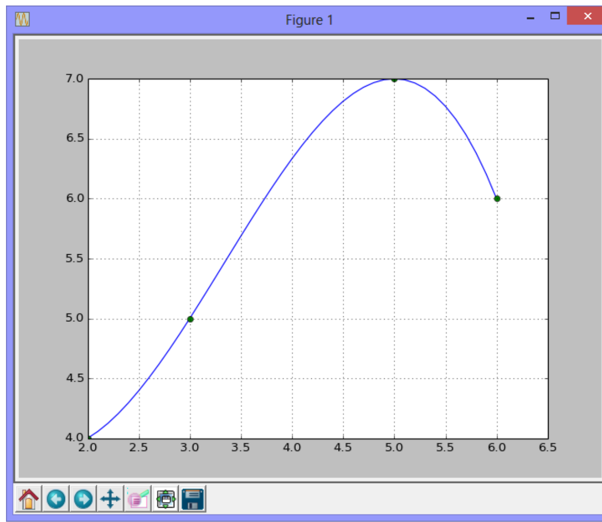
```
>>> from pylab import*
>>> from metodos import*

>>> x=[2,3,5,6]
>>> y=[4,5,7,6]
>>> p=lagrange(x,y,4)                    Evaluación del polinomio con x=4
>>> p
6.333333333333333

>>> p=lagrange(x,y)
>>> p
-t**3/6 + 5*t**2/3 - 25*t/6 + 7          Polinomio de interpolación

>>> def f(t): return -t**3/6 + 5*t**2/3 - 25*t/6 + 7

>>> t=arange(2,6.1,0.1)
>>> plot(t,f(t))                         Gráfico de los puntos y el polinomio
>>> plot(x,y,'o')
>>> grid(True)
>>> show()
```



d) Aplicación del método de Simpson

Calcular numéricamente el valor de la integral definida debajo del polinomio en el ejemplo anterior:

$$s = \int_0^2 \left(-\frac{1}{6}t^3 + \frac{5}{3}t^2 - \frac{25}{6}t + 7\right)dt$$

```
>>> from pylab import*
>>> from metodos import*

>>> def f(t): return -t**3/6 + 5*t**2/3 - 25*t/6 + 7

>>> s=simpson(f,0,2,8)                                    Con 8 franjas
>>> s
9.444444444444443
```

e) Aplicación del método de Runge-Kutta

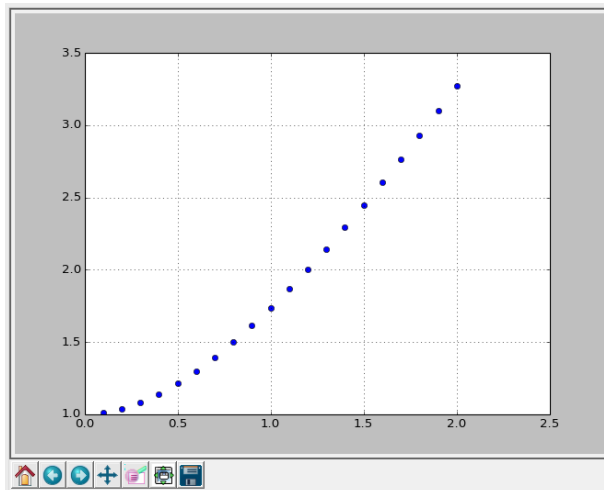
Resolver numéricamente la ecuación diferencial:

y'(x) - 2x + y - 1 = 0, 0 ≤ x ≤ 2, con la condición inicial, y(0) = 1
y'(x) = f(x,y) = 2x – y + 1

```
>>> from pylab import*
>>> from metodos import*

>>> def f(x,y): return 2*x-y+1

>>> [u,v]=rungekutta(f,0,1,0.1,20)                 Puntos de la solución y(x)
>>> plot(u,v,'o')                                  Graficar la solución numérica
>>> grid(True)
>>> show()
```



14.4 Librerías de Python para resolver numéricamente ecuaciones diferenciales

Resolver numéricamente la ecuación diferencial:

`y'(x) - 2x + y - 1 = 0,  0 ≤ x ≤ 2,` con la condición inicial, `y(0) = 1`
`y'(x) = f(x,y) = 2x - y + 1`

```
>>> from numpy import*
>>> from pylab import*
>>> from scipy.integrate import odeint

>>> def f(y,x): return 2*x-y+1                    #Note el orden de los parámetros x, y

>>> x=arange(0,2.1,0.1)
>>> y=odeint(f,1,x)
>>> plot(x,y,'o')
>>> grid(True)
>>> show()
```

x, y son vectores que contienen los puntos de la solución

x es generado con la función **arange**
y es generado con el método **odeint**

Para expresar la condición inicial: **y(0) = 1**, el primer punto del vector **x** generado con **arange** debe contener el valor de **x = 0**, mientras que el valor **y = 1** debe ser el segundo parámetro en el método **odeint**

El gráfico de la solución obtenida es idéntico al que se obtuvo con el método Runge-Kutta.

14.5 Problemas de aplicación de los métodos numéricos

1. Se necesita construir un recipiente rectangular, sin tapa, de un litro de capacidad. Para construirlo se debe usar una lámina rectangular de **32** cm de largo y **24** cm de ancho. El procedimiento será recortar un cuadrado idéntico en cada una de las cuatro esquinas y doblar los bordes de la lámina para formar el recipiente. Determine la medida del lado del cuadrado que se debe recortar en cada esquina para que el recipiente tenga la capacidad requerida. Formule el modelo matemático y resuélvalo con el método de la Bosección.

2. Un comerciante compra tres productos: A, B, C. Estos productos se venden por peso en Kg. pero en las facturas únicamente consta el total que debe pagar. El valor incluye el impuesto a las ventas y supondremos, por simplicidad que es 10%. El comerciante desea conocer el precio unitario de cada artículo, para lo cual dispone de tres facturas con los siguientes datos:

Factura	Kg. de A	Kg. de B	Kg. de C	Valor pagado
1	4	2	5	$19.80
2	2	5	8	$30.03
3	2	4	3	$17.82

Formule el modelo matemático para encontrar la solución y obténgala mediante el método de Gauss-Jordan

3. Los siguientes datos pertenecen a la curva de Lorentz, la cual relaciona el porcentaje de ingreso económico global de la población en función del porcentaje de la población:

% de población	% de ingreso global
25	**10**
50	**25**
75	**70**
100	**100**

Ejemplo. El 25% de la población tiene el 10% del ingreso económico global.

Use el método de Lagrange para obtener y graficar el polinomio de interpolacion que es el modelo para representar los datos.

4. En el techado de las casas se utilizan planchas corrugadas con perfil ondulado. Cada onda tiene la forma **f(x) = sen(x),** con un periodo de **2π** pulgadas

El perfil de la plancha tiene **8** ondas y la longitud $\mathbf{L}$ de cada onda se la puede calcular con la siguiente integral: $\mathbf{L = \int_0^{2\pi} \sqrt{1 + (f'(x))^2}\, dx}$

Use el método de Simpson con m = 4, 6, 8, 10 para calcular $\mathbf{L}$. Estime el error en el último resultado y con él, encuentre la longitud del perfil de la plancha.

5. Obtenga y grafique 10 puntos de la solución de la siguiente ecuación diferencial no lineal utilizando el método de Runge-Kutta.

$$y' - 2x + 2y^2 + 3 = 0,\quad y(0) = 1,\quad 0 \le x \le 1$$

15 Bibliografía

[1] Van Rossum, G., ***Python 3.4.1 Documentation***, Shell de Python, 2014

[2] Dowley A., ***Aprenda a pensar como un programador con Python***, 2002

[3] Bahit, E., ***Python para principiantes***, 2012

[4] Gonzáles, R., ***Python para todos***, 2012

[5] Marzal, A., ***Introducción a la programación con Python***, 2003

[6] Rougier, N., ***Matplotlib tutorial***, 2013

[7] Johansson, J., ***Sympy – Cálculo simbólico en Python***, 2014

[8] Wentworth, P., ***How to think like a computer scientist***, 2011

[9] Rodríguez, L., ***Matlab Programación***, 2013

[10] Rodríguez, L., ***Análisis Numérico Básico***, 2012

Todas las referencias y otras consultadas pero que no se han listado, fueron tomadas de la red internet en donde están disponibles bajo licencias que permiten libre acceso para uso no comercial de las mismas. Estas licencias están rotuladas como:

Creative Commons Atribución No Comercial
GNU Free Documentation License

La mayoría de los ejemplos y ejercicios propuestos en este documento fueron tomados y adaptados del libro digital ***Matlab Programación*** del mismo autor de esta obra.